СЕРИЯ

УЧЕБНОЕ ПОСОБИЕ

ПИТЕР®

УЧЕБНОЕ / ПОСОБИЕ

А. Н. Степанов

АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ И КОМПЬЮТЕРНЫХ СЕТЕЙ

Рекомендовано Министерством образования и науки Российской Федерации в качестве учебного пособия для студентов высших учебных заведений, обучающихся по специальности «Математическое обеспечение и администрирование информационных систем»

Допущено Учебно-методическим советом по прикладной математике и информатике УМО по классическому университетскому образованию в качестве учебного пособия для студентов высших учебных заведений, обучающихся по специальности «Прикладная математика и информатика» (010200) и по направлению «Прикладная математика и информатика» (510200)

Издательская программа

300 лучших учебников для высшей школы

осуществляется при поддержке Министерства образования и науки РФ

ПИТЕР®

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск
2007

ББК 32.973.202я7
УДК 004.7(075)
С79

Рецензенты:

А. Н. Терехов, доктор физико-математических наук, профессор, заведующий кафедрой системного программирования Санкт-Петербургского государственного университета;

В. А. Костин, кандидат физико-математических наук, доцент кафедры информатики Санкт-Петербургского государственного университета;

кафедра автоматизации систем вычислительных комплексов Московского государственного университета, заведующий кафедрой Л. Н. Королёв, профессор, доктор физико-математических наук, член-корреспондент РАН.

Степанов А. Н.

С79 Архитектура вычислительных систем и компьютерных сетей. — СПб.: Питер, 2007. — 509 с.: ил.

ISBN 978-5-469-01451-5
5-469-01451-7

Систематически излагаются базовые понятия и основные принципы построения архитектур вычислительных систем и компьютерных сетей, начиная от разрядно-последовательной архитектуры и заканчивая многоядерными процессорами.

Содержание учебника соответствует требованиям Государственного образовательного стандарта по специальности «Математическое обеспечение и администрирование информационных систем» (010503) и типовой программе дисциплины «Архитектура вычислительных систем и компьютерных сетей», одобренной Научно-методическим советом УМО по образованию в области математического обеспечения и администрирования информационных систем.

Материал учебника может быть полезен также студентам, обучающимся по специальности «Прикладная математика и информатика» (010501) и другим специальностям высших учебных заведений, связанным с современными информационными технологиями.

ББК 32.973.202я7
УДК 004.7(075)

ISBN 978-5-469-01451-5

Краткое содержание

Часть III. Введение в архитектуру компьютерных сетей

Содержание

Часть II. Архитектура вычислительных систем

Светлой памяти моих родителей Антонины Гавриловны и Николая Яковлевича посвящаю

Предисловие

Понятия архитектуры вычислительной системы и архитектуры компьютера являются относительно новыми. Примерно до конца 70-х гг. XX в. устройство компьютеров в высшей школе изучалось в рамках дисциплины «ЭВМ и программирование», в связи с чем и учебная литература по этим вопросам практически отсутствовала. По-видимому, первым отдельным учебником для студентов II–III курсов университетов, специализирующихся в области системного программирования, была книга Л. Н. Королева «Структуры ЭВМ и их математическое обеспечение», первое издание которой вышло в 1975 г. в издательстве «Наука» [20].

К настоящему времени в учебных планах многих специальностей высших учебных заведений, связанных с информационными технологиями, появились базовые курсы, посвященные целенаправленному изучению архитектуры компьютера. В частности, дисциплина «Архитектура вычислительных систем и компьютерных сетей» предусмотрена в учебном плане специальности 010503 «Математическое обеспечение и администрирование информационных систем».

В связи с появлением учебных дисциплин, изучающих архитектуру вычислительных систем, возникла проблема обеспеченности соответствующей учебной литературой. Вообще говоря, вопросам архитектуры вычислительных систем посвящено огромное количество специальной литературы, монографий и научных статей. Однако подавляющее большинство этих изданий рассчитано на специалистов или описывают только конкретную вычислительную систему или архитектуру. В качестве примеров можно упомянуть, например, следующие издания: Головкин Б. А. «Параллельные вычислительные системы» [12], Морс С. П., Алберт Д. Д. «Архитектура микропроцессора» 80286 [25], Григорьев В. Л. «Микропроцессор i486. Архитектура и программирование» [13], Гук М. Ю. «Аппаратные средства IBM PC» [14]. Одно из немногих изданий, посвященных общим вопросам архитектуры вычислительных машин, — монография известного американского специалиста Г. Майерса «Архитектура современных ЭВМ», перевод которой вышел в 1985 г. в издательстве «Мир» [24].

Фундаментальными руководствами по данному вопросу являются выдержавшие уже четыре издания учебники Э. Таненбаума «Архитектура компьютера» [30]

и «Компьютерные сети» [31]. Кроме того, отличным учебником по вопросам, связанным с параллельными архитектурами, может считаться книга Воеводина В. В. и Воеводина Вл. В. «Параллельные вычисления» [11], а по вопросам архитектуры компьютерных сетей — книга Олифера В. Г. и Олифера Н. А. «Компьютерные сети» [26]. Можно упомянуть также вышедшие совсем недавно издания: Барановская Т. П. и др. «Архитектура компьютерных систем и сетей» [6] и Богданова А. В. и др. «Архитектуры и топологии многопроцессорных вычислительных систем» [8], которые в целом ориентированы на обсуждение параллельных архитектур.

Много материалов по архитектурам компьютеров можно найти в периодических изданиях [1]–[4] и на Интернет-сайтах [1]–[17]. Однако для систематического изучения архитектуры компьютеров их использовать сложно. Как правило, они посвящены сиюминутным проблемам и рассчитаны на специалистов. К сожалению, большинство терминов и понятий даются без объяснения, авторы грешат сленговыми оборотами, приводимые ими сведения не дают целостного представления о предмете.

Вопрос издания учебной литературы по архитектуре компьютера, специально ориентированной на студентов математических специальностей, связанных с информационными технологиями, — таких как «Математическое обеспечение и администрирование информационных систем», «Прикладная математика и информатика», — остается весьма актуальным.

Вниманию студентов, обучающихся по указанным и родственным специальностям вузов, предлагается настоящее издание, в котором систематически вводятся необходимые базовые понятия, а также обсуждаются основные принципы построения архитектур вычислительных систем и компьютерных сетей, начиная от разрядно-последовательной архитектуры и заканчивая многоядерными процессорами.

Часть I учебника охватывает темы, которые являются вводными для изучения остального материала. В частности, в ней обсуждаются вопросы представления данных в памяти компьютера. Основное внимание уделяется числовым форматам данных, в том числе форматам вещественных чисел, описанным в стандарте IEEE 714. В первую часть также входит глава, посвященная изучению логики работы основных устройств компьютера на уровне вентилей. Однако основное внимание здесь уделено обсуждению системы машинных команд, принципа работы процессора и остальных устройств компьютера во время выполнения программы, основам низкоуровневого программирования.

Базовым вопросам архитектуры компьютера отведена часть II. Обсуждается развитие принципов параллелизма от первых компьютерных систем до настоящего времени. Здесь также изучается многопрограммный режим работы компьютеров, обсуждаются архитектурные приемы повышения эффективности оперативной памяти, кэша и шинной системы компьютера. Рассматриваются суперконвейерная и суперскалярная архитектуры современных компьютеров, а также комплекс вопросов, связанных с динамическим исполнением машинных команд — предсказание переходов, изменение последовательности выполнения команд, спекуля-

тивное выполнение. Обсуждаются также вопросы многопоточного исполнения (Hyper Threading) в архитектуре IA-32 и предикатного выполнения машинных команд в архитектуре IA-64. Отдельные главы посвящены обсуждению архитектурных особенностей классов CISC, RISC, VLIW, EPIC и некоторых популярных семейств (Intel, AMD, SUN SPARC), параллельным и неклассическим архитектурам.

В части III кратко обсуждаются вопросы, связанные с архитектурой компьютерных сетей. Рассматриваются основные методы доступа и схемы коммутации, а также базовые сетевые технологии и эталонная модель OSI.

Содержание учебника соответствует требованиям государственного образовательного стандарта по специальности 010503 «Математическое обеспечение и администрирование информационных систем» и типовой программе дисциплины «Архитектура вычислительных систем и компьютерных сетей», одобренной научно-методическим советом УМО по образованию в области конфликтологии, искусств и гуманитарных наук, математического обеспечения и администрирования информационных систем, а также рекомендациям «Computing Curricula 2001: Computer Science», раздел AR (Рекомендации по преподаванию информатики в университетах / Пер. с англ. СПб.: Изд-во СПбГУ, 2002 [27]), и рекомендациям «Software Engineering 2004. Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering». Материал учебника может быть полезен также студентам специальности 010501 «Прикладная математика и информатика» и других специальностей высших учебных заведений, связанных с современными информационными технологиями.

Для изучения архитектуры вычислительных систем и компьютерных сетей студент должен освоить информатику и дискретную математику в объеме стандартного университетского курса указанных ранее математических специальностей. Понятия и принципы, изучающиеся в курсе, используются в дальнейшем при изучении операционных систем и оболочек, баз данных и СУБД, администрирования информационных систем, низкоуровневых языков программирования.

В тексте учебника приняты следующие условные обозначения.

Новые понятия и термины в предложениях, в которых они встречаются впервые, а также в предложениях, в которых они определяются, выделены **полужирным** шрифтом.

ВНИМАНИЕ

Врезки «Внимание» содержат определения или объяснения каких-либо понятий и терминов или указания, которые требуют особого внимания читателя.

ПРИМЕЧАНИЕ

Примечания содержат сведения, полезные для желающих более детально изучить тот или иной вопрос, но не являющиеся обязательными для освоения курса.

СОВЕТ

Кроме того, в книге встречаются некоторые советы.

Благодарности

Прежде всего я хочу выразить свою глубокую признательность ректору Самарского государственного университета профессору Геннадию Петровичу Яровому, без существенной поддержки которого учебник не был бы написан.

Я благодарен моим рецензентам члену-корреспонденту РАН Л. Н. Королеву, профессорам А. Н. Терехову, Б. А. Новикову и доценту В. А. Костину за ценные советы и замечания, позволившие устранить неточности и улучшить качество этой книги.

Особую признательность хотелось бы высказать В. Ю. Шачину, Н. В. Рощиной, К. А. Кнопу, а также всем сотрудникам издательства «Питер», которые принимали участие в подготовке и выпуске этого учебника.

От издательства

Ваши замечания, предложения и вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер»).

Мы будем рады узнать ваше мнение!

Подробную информацию о наших книгах вы найдете на веб-сайте издательства http://www.piter.com.

Часть I
Введение в архитектуру компьютера

В первую часть включены вопросы, которые, вообще говоря, являются вводными для изучения остального материала учебника. В частности, обсуждаются некоторые исходные понятия архитектуры компьютера, способы представления данных и программ в компьютере, а также логика работы его основных устройств на физическом уровне. Некоторые из этих вопросов частично излагаются в общеобразовательном курсе информатики либо относятся к пограничной области, но для связности и полноты изложения материала они также включены в настоящий учебник. Предполагается, что читатель знаком с основными понятиями систем счисления, с особенностями двоичной и шестнадцатеричной систем счисления, с переходами между ними и выполнением арифметических операций в этих системах.

Глава 1
Начальные сведения по устройству компьютера

Компьютер представляет собой универсальное устройство, которое используется для автоматической обработки информации, осуществляемой по заранее составленному плану. Такой план, обладающий необходимыми для получения требуемого результата свойствами, принято называть **алгоритмом**, а его запись в определенной, «понятной» компьютеру форме называется **программой**.

В отечественной литературе до 1985 г. в основном использовались термин **электронная вычислительная машина** и его аббревиатура **ЭВМ**, часто встречался также оборот **вычислительная машина**. После 1985 г. широкое распространение получил англоязычный термин «компьютер» (от computer — вычислитель). Мы будем использовать эти термины в основном как равноправные. Следует, однако, отметить, что в последнее время активно ведутся разработки компьютеров, работа которых основана на оптических, квантовых и некоторых других физических принципах. В связи с этим понятие «электронная вычислительная машина», в котором акцентируется, что машина построена на основе электронных устройств, становится более узким, чем понятие «компьютер».

ВНИМАНИЕ

Компьютер — это универсальное устройство, используемое для автоматизации процессов приема, хранения, обработки и передачи информации, которые осуществляются по заранее разработанным человеком программам.

Обращаем внимание на важнейшие моменты этого определения:

1) компьютер представляет собой *устройство* (обычно электронное);
2) компьютер выполняет действия без вмешательства человека — *автоматически*;
3) для этого ему должен быть заранее передан разработанный человеком и записанный в специальной форме план действий — *программа*;

4) это устройство является *универсальным* в том смысле, что оно может выполнять *любую* обработку информации, для которой имеется соответствующая программа.

Существует множество автоматических устройств, которые так или иначе обрабатывают информацию. Это, например, всевозможные автоматы по продаже товаров, банкоматы (банковские автоматы), телефоны-автоматы, встроенные в бытовую технику устройства управления и т. д., но в подавляющем большинстве они имеют узкую специализацию и не могут выполнить произвольно заданную программу — именно в этом их коренное отличие от компьютеров, являющихся *универсальными устройствами обработки информации.*

Подчеркнем, что аппаратура компьютера принципиально не может выполнять никаких действий без программы, описывающей эти действия. Аппаратура компьютера без программы подобна автомобилю без водителя. Но и программы сами по себе без аппаратуры не могут обработать данные, так же как водитель без автомобиля не может перевести пассажиров или груз. Таким образом, аппаратура компьютера и выполняемые программы образуют *систему.*

ВНИМАНИЕ

Совокупность устройств одного или нескольких компьютеров принято называть аппаратным обеспечением, или аппаратными ресурсами. Совокупность программ, которые могут быть выполнены на этих устройствах, называется программным обеспечением, или программными ресурсами. Вычислительной системой называется комплекс, который включает в себя совокупность взаимодействующих в процессе обработки данных аппаратного и программного обеспечения.

Напомним, что *системой* называется сложная структура, состоящая из взаимодействующих компонентов, каждый из которых в отдельности не обладает свойствами, присущими системе в целом. Свойства, которые присущи системе в целом и не присущи никакому ее компоненту, называют *системными свойствами.* В данном случае системным свойством является способность вычислительной системы обрабатывать данные. Вычислительная система — это общее понятие. Компьютер (вместе с программами) является частным случаем вычислительной системы.

1.1. Данные и программы

Из курса информатики известно, что **информация** представляет собой нематериальное содержание, которое извлекается человеком из некоторого материального сообщения. И следовательно, вести речь об обработке информации каким-либо «неодушевленным» устройством в принципе невозможно. Следует говорить об обработке сообщений, которая осуществляется такими устройствами. Но в силу укоренившейся традиции устной и письменной речи существенные различия между сообщением и информацией игнорируются, и даже в учебной и научной литературе почти всегда говорят о передаче информации, обработке информации, хранении информации и т. д.

Носителем сообщения называется любая материальная среда, служащая для его хранения или передачи. Примеры носителей сообщений: бумага, воздух, электромагнитные колебания, электронные схемы, магнитные и оптические диски и т. д.

В общем случае сообщение — это последовательность зафиксированных каким-либо образом сигналов. **Сигнал** представляет собой изменение во времени или в пространстве материального объекта — носителя сообщения или же некоторой его характеристики, при этом сама изменяющаяся характеристика называется **параметром сигнала**. Важными для практического использования примерами сигналов могут служить изменения амплитуды, фазы или частоты звуковых, а также электромагнитных колебаний. При этом амплитуда, фаза или частота являются возможными параметрами сигнала.

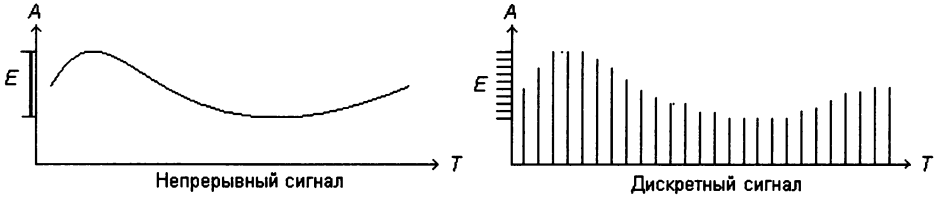


Рис. 1.1. Непрерывный и дискретный сигналы

Сигнал называется **непрерывным**, или **аналоговым**, если множество значений его параметра бесконечно — точнее, более чем счетно. Например, звуковые и электромагнитные сигналы относятся к непрерывным, так как их параметры могут принимать любые значения из некоторого отрезка числовой оси. На рис. 1.1, *слева*, изображен пример непрерывного сигнала, параметр которого *A* может принимать любые значения из интервала *E*.

Сигнал называется **дискретным**, если его параметр может принимать лишь конечное число значений. На рис. 1.1, *справа*, изображен пример дискретного сигнала, параметр которого *A* может принимать только 10 различных значений, образующих конечное множество *E*. Эти значения изображены короткими горизонтальными штрихами на оси ординат. Дискретные сигналы используются в любых письменных текстах, в разнообразных устройствах — часах, информационных табло и т. д., в том числе в компьютерах. Отметим, что теоретически допускаются дискретные сигналы, имеющие счетное количество различных значений параметра, но на практике важны только сигналы с конечными наборами значений параметров.

Сообщения, основанные на непрерывных сигналах, называются непрерывными, а сообщения, основанные на дискретных сигналах, — дискретными. В то же время информация, в отличие от сообщений, не обладает ни свойством непрерывности, ни свойством дискретности.

Знаком называется элемент некоторого множества объектов, используемых для хранения или передачи дискретных сообщений. С обсуждавшейся ранее точки зрения, знак — это одно из возможных значений параметра дискретного сигнала. Знак вместе с его смыслом принято называть **символом**. Множество знаков, в ко-

тором определен линейный порядок, называется **алфавитом**. Для практических нужд человек использует множество различных алфавитов: алфавиты естественных языков, условные знаки на чертежах и топографических картах, знаки дорожного движения, нотные знаки и т. д. Для передачи или хранения сообщений иногда используются множества знаков, в которых отсутствует упорядоченность, например множество иероглифов китайского языка, набор знаков азбуки Брайля (специальной азбуки для восприятия текстов слепыми) и т. д.

Из теории кодирования информации известно, что при соблюдении некоторых условий сообщение, заданное в одном алфавите, можно без потери его смысла (без потери информации) задать в другом алфавите. Это создает возможность свести все разнообразие используемых человечеством алфавитов к одному, наиболее удобному в некотором смысле алфавиту. Теоретически и экспериментально было показано, что самым удобным и эффективным для представления сообщений в компьютерах является использование двоичных алфавитов, в которых фигурируют всего два различных знака.

ПРИМЕЧАНИЕ

Можно показать (см., например, [29]), что оптимальным является алфавит, содержащий ближайшее целое к e = 2,71828... количество знаков (e — основание натуральных логарифмов). Очевидно, что это число 3, а не 2, и следовательно, более эффективным является использование троичного алфавита. Однако технические трудности в реализации компьютера, базирующегося на троичном алфавите, оказались значительными, поэтому современные компьютеры используют для представления информации исключительно двоичный алфавит.

Примеры двоичных алфавитов: {0, 1}, {true, false}, пара напряжений {1 В, 5 В} и т. д. Отметим, что используемые в двоичном алфавите {0, 1} знаки 0 и 1 часто называют также **двоичными цифрами**.

Когда говорят об обработке информации с помощью компьютеров, обычно в качестве термина, эквивалентного термину «информация» (следует понимать, что речь идет о «сообщениях»), используют термин «данные». Отметим, что план обработки данных — **алгоритм** — с общей точки зрения, *также является сообщением*, которое записано в некотором алфавите. А если этот алфавит и правила задания алгоритма воспринимаются вычислительной системой, тогда применяется термин «программа».

ВНИМАНИЕ

Обрабатываемую информацию, записанную в «понятной» компьютеру форме, принято называть данными. План обработки данных — алгоритм, также записанный в специальной, «понятной» компьютеру форме, — принято называть программой.

Как следует из сказанного ранее, естественной, «понятной» для компьютера формой задания алгоритмов и обрабатываемых данных является их запись в двоичном алфавите {0, 1}. Переход к двоичному алфавиту в записи программ и данных принято называть **двоичным кодированием**. А сами программы и данные,

записанные в этом алфавите, часто называют **двоичным кодом**. Поскольку двоичный код используется для хранения информации в компьютерах — вычислительных машинах, его называют также **машинным кодом**.

1.2. Понятие архитектуры компьютера

Термин «архитектура» в применении к компьютерам относительно нов. Почти до конца 70-х гг. применялись более узкие термины «устройство» или «структура» компьютера [20]. Но, как правило, после рассмотрения собственно устройства машины следовало изучение способов представления программ и данных в компьютере, особенностей различных устройств компьютера, организации обмена данными и т. д. Впоследствии всю эту совокупность сведений объединили под одним общим названием — **архитектура компьютера**.

ВНИМАНИЕ

Под архитектурой компьютера понимается совокупность взаимосвязанных сведений о способах представления в компьютере программ и данных, о назначении, структуре и особенностях функционирования отдельных его устройств, а также об организации и работе компьютера в целом.

Из сказанного следует, что основными функциями компьютера являются хранение, обработка, прием и передача данных. Для выполнения каждой из этих функций в компьютере предусмотрены специальные устройства:

- **память** — группа устройств, которые обеспечивают *хранение* программ и данных;
- **процессор** (от process — обработка) — одно или несколько устройств, которые обеспечивают задаваемую программой *обработку* данных;
- **устройства ввода/вывода** — группа устройств, которые обеспечивают обмен, то есть *прием и передачу* данных между пользователем и компьютером или между двумя или более компьютерами.

Различные устройства компьютера подсоединяют друг к другу с помощью стандартизированных и унифицированных (то есть единообразных) аппаратных средств — кабелей, разъемов и т. д. При этом устройства обмениваются друг с другом информацией и управляющими сигналами, которые также приводятся к некоторым стандартным формам. Совокупность этих стандартных средств и форм образует конкретный **интерфейс** (от interface — взаимный вид) того или иного устройства или программы.

ВНИМАНИЕ

Интерфейсом называется совокупность унифицированных стандартных соглашений, аппаратных и программных средств, методов и правил взаимодействия устройств или программ, а также устройств или программ с пользователем.

Изучение архитектуры компьютера естественно начать с обсуждения логического строения его памяти.

1.3. Элементарные логические устройства памяти

Память компьютера имеет сложную многоуровневую структуру, реализованную в виде взаимодействующих устройств, которые могут использовать различные физические принципы для хранения данных. К этим устройствам относятся интегральные схемы, магнитные и оптические диски и т. д. Многоуровневый подход к реализации памяти вытекает из необходимости обеспечения эффективной работы компьютера при решении задач, точнее, при выполнении соответствующих им программ. Но в любом случае, при любой физической реализации памяти ее базовыми функциональными элементами являются **бит** и **байт**.

ВНИМАНИЕ

Элементарное устройство памяти компьютера, которое применяется для хранения одного из знаков двоичного алфавита, называется битом, или двоичным разрядом.

Термин «бит» произошел от английского bit, представляющего собой сокращение словосочетания BInary digiT — двоичная цифра (в некоторых источниках BInary elemenT — двоичный элемент). Способы физической реализации битов памяти в компьютерах обсуждаются в дальнейшем, а пока нас будут интересовать только возможности, точнее, функции бита.

- Бит может находиться только в одном из двух возможных устойчивых состояний, одно из которых принято считать изображением знака «0», а другое — изображением знака «1». Свое состояние бит сохраняет сколь угодно долго, пока оно не будет изменено принудительным способом. Следовательно, бит может хранить записанную в нем информацию.
- В любой момент времени можно узнать, в каком из двух состояний находится бит, — в состоянии «0» или в состоянии «1»; при этом текущее состояние бита останется неизменным. Другими словами, можно прочитать записанную в бит информацию (без ее потери).
- Всегда, когда в этом возникнет необходимость и вне зависимости от текущего состояния, можно перевести бит из одного состояния в другое. Иначе говоря, в бит можно записать новую информацию.

Итак, бит обеспечивает необходимую основу для реализации одной из важнейших функций компьютера — *хранения* данных.

Бит — это очень маленькая порция данных. Поэтому как для записи десятичных чисел используется несколько десятичных разрядов — разряд единиц, разряд десятков, сотен и т. д., так и для хранения двоичных машинных кодов используется несколько битов, совместно образующих устройство, которое принято называть **ячейкой памяти**. В общем случае ячейки различных компьютеров могут состоять из различного количества битов. Однако это создает значительные сложности для организации обмена данными между разными моделями компьютеров. Поэтому, начиная с машин третьего поколения, стандартными являются ячейки, состоящие из восьми битов.

ВНИМАНИЕ

Самостоятельный элемент памяти компьютера, состоящий из восьми битов, называется байтом.

Слово «байт» произошло от английского термина byte, представляющего собой сокращение словосочетания BinarY TErm — двоичный терм, двоичное выражение. Байт сохраняет все перечисленные выше свойства бита — он может хранить записанную в него информацию, ее можно прочитать, можно также записать в байт любую новую информацию.

Более подробно обсудим, что представляет собой содержимое байта. Каждый из восьми битов байта может содержать любую из двоичных цифр *независимо* от остальных. Следовательно, байт может содержать *произвольную комбинацию*, последовательность из восьми нулей или единиц, например последовательность 10110011. Такую последовательность называют также двоичным числом, или двоичным кодом. Количество различных кодов, различных комбинаций из восьми нулей и единиц, записываемых в один байт, равно $2^8 = 256$.

Запись находящегося в байте двоичного кода можно спутать с аналогичным по записи десятичным числом. Например, двоичный код 10110011 можно рассматривать и как «обычное» число «десять миллионов сто десять тысяч одиннадцать». В тех случаях, когда есть опасность спутать десятичное число и двоичный код, справа от двоичного кода записывают нижний индекс 2, а справа от десятичного числа указывают индекс 10. Таким образом, 10110011_2 — двоичный код, а 10110011_{10} — десятичное число. Для удобства восприятия десятичные числа в текстах на русском языке принято делить на группы по три цифры в каждой и отделять эти группы друг от друга пробелом — $10\ 110\ 011_{10}$. По аналогии с этим двоичные коды иногда также группируют, но по четыре цифры в группе — $1011\ 0011_2$.

Все байты одной из важнейших разновидностей памяти — оперативной — пронумерованы целыми числами. При этом номером начального байта всегда считается ноль. С помощью номера легко отличить один байт от другого, в любой момент можно «открыть» байт с нужным номером и посмотреть, какой код в нем находится, либо заменить его любым другим.

ВНИМАНИЕ

Байты, в отличие от отдельных битов, из которых они состоят, являются самостоятельными элементами памяти компьютера, так как именно с байтом в целом, а не с отдельным битом производятся такие операции, как запись и чтение.

Условно бит изображают в виде небольшого прямоугольника, содержащего либо цифру 0, либо цифру 1 (рис. 1.2, *а*), а байт рисуют в виде расположенных рядом восьми одинаковых прямоугольников, каждый из которых содержит какую-либо двоичную цифру. Биты, входящие в байт, принято нумеровать справа налево, начиная с нуля, — так, как показано на рис. 1.2, *б*.

Во многих случаях для записи некоторой неделимой порции информации, например для записи кода какого-либо числа восьми разрядов, одного байта не хватает. Тогда несколько соседних, смежных байтов объединяются в структуру,

которую принято называть *полем*. Соседними считаются байты, номера которых образуют арифметическую прогрессию n, $n + 1$, $n + 2$, ..., $n + k$. При этом значение выражения $k + 1$ считается длиной поля.

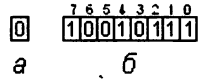


Рис. 1.2. Условные изображения: *а* — бита; *б* — байта

ВНИМАНИЕ

Группа смежных байтов памяти, которая используется для записи неделимой порции информации или для каких-либо других целей, называется полем. Количество байтов, из которых состоит поле, называется длиной поля.

Как следует из определения, в общем случае в полях памяти могут находиться и логически не связанные между собой данные.

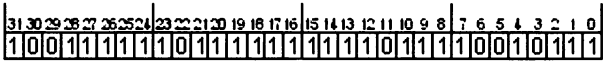


Рис. 1.3. Нумерация разрядов в поле памяти

Когда байты памяти рассматриваются отдельно друг от друга, нумерация битов в каждом из них производится независимым образом, как показано на рис. 1.2, *б*. Если же байты входят в поле, то применяется сквозная нумерация битов по всему полю. На рис. 1.3 приведен пример нумерации битов поля длиной четыре байта. Высокие линии отмечают границы отдельных байтов, входящих в поле.

1.4. Объем памяти

Важнейшей характеристикой любых устройств памяти компьютера является **объем**. Объем памяти равен количеству байтов, из которых она состоит. Заметим, что объем памяти, который измеряется в байтах, не имеет ничего общего с физической величиной «объем тела», которая измеряется в литрах, кубических сантиметрах, кубических метрах и т. д. Это величина не кубической, а линейной природы. Она аналогична длине тела, измеряемой в сантиметрах, метрах и т. д. Впрочем, термин «объем» используется, когда говорят о памяти устройства в целом. Ранее установлено, что когда речь идет о характеристике некоторого участка (например, поля памяти), также используется термин «длина». Следовательно, *объем памяти* и *длина участка памяти* представляют собой одну и ту же характеристику — количество байт, из которых состоит обсуждаемый объект. В дальнейшем изложении эти термины используются как эквивалентные.

Байт является основной единицей измерения объема памяти. Вместе с тем байт как единица измерения представляет собой слишком маленькую величину. Поэтому для указания объемов памяти используется целый ряд кратных единиц, которые образуются с помощью так называемой **двоичной тысячи**, равной $2^{10} = 1024$.

Первая кратная единица называется **килобайт** (Кбайт, произносится «кабайт»): 1 Кбайт = 1024 байт. Собственно говоря, называть эту единицу килобайтом не совсем правильно, так как килобайт должен быть равным 1000 байт, а не 1024. Но обычно этой небольшой разницей пренебрегают. Следующая кратная единица — **мегабайт** (Мбайт): 1 Мбайт = 1024 Кбайт = 1 048 576 байт. Далее следуют **гигабайт** (Гбайт, около миллиарда байт) и **терабайт** (Тбайт, около триллиона байт). Можно упомянуть и еще более крупные единицы: **петабайт** (Пбайт), **эксабайт** (Эбайт), **зетабайт** (Збайт) и **йоттабайт** (Йбайт). Для наглядности значения кратных единиц сведены в табл. 1.1.

Таблица 1.1. Кратные единицы объема памяти

Единица	Значение	Метрический аналог
1 Кбайт	1024 байт (2^{10})	1000 (10^3)
1 Мбайт	1024 Кбайт = 1 048 576 байт (2^{20})	1 000 000 (10^6)
1 Гбайт	1024 Мбайт = 1 073 741 824 байт (2^{30})	10^9
1 Тбайт	1024 Гбайт = 1 099 511 697 776 байт (2^{40})	10^{12}
1 Пбайт	1024 Тбайт = 125 899 978 522 624 байт (2^{50})	10^{15}
1 Эбайт	1024 Пбайт = 1 152 921 504 606 846 976 байт (2^{60})	10^{18}
1 Збайт	1024 Эбайт = 1 180 591 620 717 411 303 424 байт (2^{70})	10^{21}
1 Йбайт	1021 Збайт = 1 208 925 819 614 629 174 706 176 байт (2^{80})	10^{24}

Контрольные вопросы и упражнения

1. Чем отличается компьютер от других устройств, обеспечивающих обработку информации?
2. Чем отличается электронно-вычислительная машина от компьютера?
3. Что называется вычислительной системой?
4. Дайте определения понятий «аппаратные ресурсы» и «программные ресурсы».
5. Что представляет собой сообщение? Чем отличается сообщение от информации?
6. Что может быть использовано в качестве носителя сообщений?
7. Приведите примеры сигналов и их возможных параметров. Чем отличается непрерывный сигнал от дискретного?
8. Что представляет собой дискретное сообщение?
9. Поясните смысл терминов «знак», «символ», «алфавит».
10. Что понимается под термином «данные»?
11. Почему в компьютерах для представления данных и программ используется двоичная система счисления?
12. Что представляет собой машинный код?

13. Дайте определение понятия «архитектура компьютера».
14. Перечислите основные группы устройств компьютера и охарактеризуйте их назначение.
15. Что называется интерфейсом?
16. Что представляет собой бит? Перечислите его функции. Как он может быть реализован?
17. Что такое байт? Перечислите его функции. Чем байт отличается от ячейки?
18. Сравните между собой бит и байт. Чем они похожи и чем различаются?
19. Что такое поле? Как определяется длина поля?
20. Как нумеруются биты в байтах и в полях?
21. Дайте определения единиц измерения объема памяти.

Глава 2
Представление данных в компьютере

При компьютерной обработке информации приходится иметь дело с текстовыми, графическими, числовыми, звуковыми и другими данными. Для хранения данных различной природы применяются различные способы их представления в двоичном алфавите — различные способы кодировки. Кроме того, *для одной и той же* разновидности данных также могут использоваться *различные* способы кодировки, которые отличаются друг от друга эффективностью и различными требованиями к ресурсам компьютера.

ВНИМАНИЕ

Конкретный способ кодирования той или иной разновидности информации в компьютере принято называть *форматом данных*.

В общем случае термин «формат» понимается как строго определенный, исчерпывающе полный набор правил. Следовательно, в приведенном ранее определении речь идет об *исчерпывающем наборе правил кодирования* в компьютере той или иной разновидности данных.

2.1. Текстовые данные

При хранении в компьютере любой текст (документ, статья, книга) рассматривается как линейная последовательность символов. Причем промежуток между отдельными словами — пробел, переход на следующую строчку, переход на следующую страницу — также могут рассматриваться как некие специальные символы. Каждому символу из этой последовательности ставится в соответствие конкретный двоичный код, состоящий *ровно из восьми* двоичных разрядов. Таким образом, код каждого символа текста занимает *ровно один байт* памяти. Следовательно,

текст целиком занимает столько байтов памяти компьютера, из скольких символов он состоит (включая все его символы — пробелы, знаки препинания, специальные знаки перехода на новую строчку, на новую страницу и т. д.).

Алфавит, используемый для представления текстов на естественном языке, должен содержать как минимум 52 латинские буквы (строчные и прописные), десятичные цифры, знаки препинания, математические знаки, специальные знаки и т. д., всего примерно 150 символов. Исходя из теоретических соображений, это требует при равномерном алфавитном двоичном кодировании для представления любого знака исходного алфавита $\log_2 150 \approx 7{,}2$, то есть восьми двоичных цифр [29].

Списки всех используемых при записи текстов символов и *однозначно* соответствующих им двоичных кодов образуют так называемые **кодовые таблицы**. В практике программирования применяются различные кодовые таблицы. Одной из наиболее часто используемых является кодовая таблица **ASCII** (American Standard Code for Information Interchange — американский стандартный код для обмена информацией). В этой таблице зафиксированы коды для 128 различных символов. Их список и соответствующие им восьмиразрядные (то есть состоящие из восьми двоичных цифр, разрядов) двоичные коды образуют основную (базовую) кодовую таблицу ASCII. Но один байт может содержать 256 различных восьмиразрядных кодов. Это означает, что в стандарте ASCII задействована только половина возможностей 8-битного кодирования. Имеются различные расширения основной кодовой таблицы ASCII, в которых задаются коды еще для 128 символов, в том числе и для символов различных национальных алфавитов. Фрагмент одного из расширений кодовой таблицы ASCII, включающий буквы русского алфавита — кириллицы, приведен в табл. 2.1.

Таблица 2.1. Фрагмент кодовой таблицы

Символ	Двоичный код	Символ	Двоичный код	Символ	Двоичный код	Символ	Двоичный код
А	10000000	И	10001000	Р	10010000	Ш	10011000
Б	10000001	Й	10001001	С	10010001	Щ	10011001
В	10000010	К	10001010	Т	10010010	Ъ	10011010
Г	10000011	Л	10001011	У	10010011	Ы	10011011
Д	10000100	М	10001100	Ф	10010100	Ь	10011100
Е	10000101	Н	10001101	Х	10010101	Э	10011101
Ж	10000110	О	10001110	Ц	10010110	Ю	10011110
З	10000111	П	10001111	Ч	10010111	Я	10011111

В качестве примера кодировки получим машинный код текста, состоящего из одного слова «КОМПЬЮТЕР». Этот текст состоит из 9 символов, следовательно, для его хранения требуется 9 байтов памяти. Используя табл. 2.1, для каждого символа легко получить соответствующий ему двоичный код. Остается только записать найденные коды в группу расположенных подряд байтов памяти. В табл. 2.2

приведен полученный таким образом машинный код этого текста. В первой строке таблицы указаны порядковые номера байтов памяти, в которых записан текст, во второй — символы, из которых текст состоит, а в третьей — машинные, двоичные коды символов. Таким образом, текст «КОМПЬЮТЕР» в вычислительной машине представлен двоичным кодом 1000 1010 1000 1110 1000 1100 1000 1111 1001 1100 1000 1110 1001 0010 1000 0101 1001 0000_2. Следует понимать, что пробелы между четверками двоичных цифр вставляются только для удобства их восприятия (чтения человеком), и в память компьютера они, естественно, не записываются.

Таблица 2.2. Машинный код текста «КОМПЬЮТЕР»

1	2	3	4	5	6	7	8	9
К	О	М	П	Ь	Ю	Т	Е	Р
10001010	10001110	10001100	10001111	10011100	10001110	10010010	10000101	10010000

Обратите внимание на то, что в табл. 2.1 приведены коды прописных букв. Строчные буквы имеют другие коды. Например, код буквы «а» имеет вид 1010 0000_2, в то время как код буквы «А» — 1000 0000_2. Не случайно рассматриваемое слово записано именно в таком виде — машинный код слова «КОМПЬЮТЕР» *отличается* от машинного кода слова «компьютер».

Если учесть все возможные буквы, входящие в национальные алфавиты различных стран, все возможные символы, которые встречаются в математических и других специальных текстах, то их общее количество окажется значительно больше 256 символов, кодировку которых обеспечивает один байт[1]. Поэтому было разработано несколько десятков различных кодовых таблиц. При этом в разных кодовых таблицах один и тот же код может соответствовать разным символам. Так, например, двоичный код 1000 1010_2 соответствует символу «К» только в так называемой **ГОСТ-альтернативной** кодовой таблице. Именно ее фрагмент (она является одним из расширений базовой кодовой таблицы) приведен в табл. 2.1. А в другой популярной кодовой таблице, **Windows 1251**, этот же двоичный код служит для обозначения символа «Љ». Следовательно, текст, записанный какой-либо программой в одной кодовой таблице, может быть полностью искажен при его чтении с помощью другой программы, использующей другую кодовую таблицу. Если приведенный код слова «КОМПЬЮТЕР» попытаться прочитать с помощью программы, которая использует кодовую таблицу «Windows 1251», то он окажется соответствующим «слову» «Љђ Њђ њђ '...ђ».

Учитывая недостаточность возможностей обсуждавшихся кодовых таблиц, в последнее время все шире используется кодовая таблица с названием **UNICODE** (UNIversal CODE — универсальный код), в которой для кода одного символа отводится два байта, а не один, как в рассмотренных ранее таблицах. С помощью двух байтов, то есть 16 битов, можно закодировать уже 2^{16} = 65 536 различных

[1] По некоторым оценкам, во всем мире в настоящее время используется около 200 000 различных символов.

символов. Такого количества кодов вполне достаточно, чтобы представить большинство из встречающихся во всевозможных текстах символов.

Если внимательно проанализировать действия, которые приходится выполнять над произвольными текстами, можно заметить, что любые их преобразования сводятся к замене одного символа текста другим, удалению символа из текста и вставке нового символа в выбранное место текста. В тех случаях, когда упомянутые действия по преобразованию текстов необходимо выполнить не над одним символом, а над некоторой их группой, соответствующие операции можно выполнить циклически несколько раз. Если включить в алфавит специальный «пустой» символ (не путать с пробелом!), который можно считать совпадающим с пустым текстом (то есть текстом, не содержащим ни одного символа), то окажется, что любые преобразования текстов сводятся к одной-единственной операции: замене одного символа алфавита другим. Естественно, для определения заменяемого или удаляемого символа либо места вставки нового символа нужно еще уметь сравнивать между собой любые два символа алфавита на совпадение или несовпадение.

Итак, основные элементарные действия, которые должен «уметь» выполнять компьютер над текстовыми данными, — это сравнение кодов двух символов текста и замена одного кода другим.

2.2. Графические данные

Для обсуждения общих принципов кодирования графических данных в качестве конкретного, достаточно общего случая графического объекта выберем изображение на экране телевизора или **дисплея** (от display — показывать, демонстрировать) компьютера. Это изображение состоит из некоторого количества горизонтальных линий — строк (рис. 2.1). А каждая строка, в свою очередь, состоит из элементарных мельчайших единиц изображения — точек, которые принято называть **пикселами** (от pixel — PICture'S ELement — элемент картинки). Весь массив элементарных единиц изображения называют **растром** (от лат. rastrum — грабли). Степень четкости изображения зависит от количества строк на весь экран и количества точек в строке, которые представляют **разрешающую способность** экрана, или просто **разрешение**. Чем больше строк и точек, тем четче и лучше изображение. Достаточно хорошим считается, например, разрешение 800 × 600, то есть 800 точек на строку и 600 строчек на экран.

Строки, из которых состоит изображение, можно просматривать сверху вниз друг за другом, как бы составив из них одну сплошную линию. После полного просмотра первой строки просматривается вторая, за ней третья, потом четвертая и т. д. до последней строки экрана. Этот процесс очень похож на принятый в большинстве стран мира способ чтения текстов, когда строчки просматриваются друг за другом слева направо и сверху вниз. Такой способ работы с изображением называется **строчной разверткой**, или **сканированием** (от scan — бегло просматривать, развертывать изображение). Так как каждая из строк представляет собой последовательность пикселов, то все изображение, вытянутое в линию, также

можно считать линейной последовательностью элементарных точек. В рассматриваемом случае эта последовательность состоит из 800 × 600 = 480 000 пикселов.

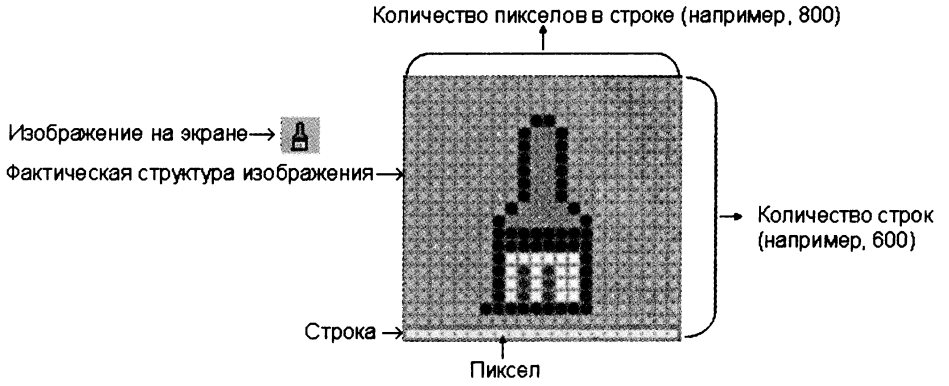


Рис 2.1. Структура растрового изображения

Вначале обсудим принципы кодирования монохромного изображения, то есть изображения, состоящего из любых двух контрастных цветов — черного и белого, зеленого и белого, коричневого и белого и т. д. Для определенности будем считать, что один из цветов — черный, а второй — белый. Тогда каждый пиксел изображения может иметь либо черный, либо белый цвет. Поставив в соответствие черному цвету двоичный код 0, а белому — код 1 (либо наоборот), мы сможем закодировать в одном бите состояние одного пиксела монохромного изображения. Так как 1 байт состоит из 8 битов, то на строчку, состоящую из 800 пикселов, потребуется 100 байтов памяти (рис. 2.2), а на все изображение — 60 000 байтов.

Рис. 2.2. Пример кодирования строки черно-белого изображения

Восстановление изображения по известному его коду может осуществляться, например, с помощью луча электронно-лучевой трубки телевизора или монитора компьютера, который в процессе сканирования строк экрана под управлением специальной декодирующей схемы создает соответствующий коду цвет пиксела (белый или черный).

Однако полученное таким образом изображение будет чрезмерно контрастным. Реальное черно-белое изображение состоит не только из белого и черного цветов. В него входит множество различных промежуточных оттенков — серый, светло-серый, темно-серый и т. д. Если кроме белого и черного цветов использовать только две дополнительные градации, скажем, светло-серый и темно-серый цвета, то чтобы закодировать цветовое состояние одного пиксела, потребуется уже *два* бита. При этом кодировка может быть, например, такой: черный цвет — 00_2, темно-серый — 01_2, светло-серый — 10_2, белый — 11_2 (рис. 2.3).

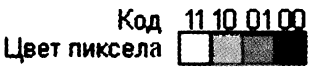


Рис. 2.3. Вариант кодирования промежуточных оттенков монохромного изображения

Общепринятым на сегодняшний день и дающим достаточно реалистичные монохромные изображения считается кодирование состояния одного пиксела с помощью восьми битов, то есть одного байта, которое позволяет передавать 256 различных оттенков серого цвета, от полностью белого до полностью черного. В этом случае для передачи всего растра из 800 × 600 пикселов потребуется уже не 60 000, а 480 000 байтов.

Цветное изображение может формироваться различными способами. Один из них — метод **RGB** (Red, Green, Blue — красный, зеленый, синий), который опирается на то, что глаз человека воспринимает все цвета как сумму трех основных — красного, зеленого и синего. Например, сиреневый цвет — это сумма красного и синего, желтый цвет — сумма красного и зеленого и т. д. Для получения цветного пиксела в одну и ту же точку экрана направляется не один белый, а сразу три цветных луча. Опять упрощая ситуацию, будем считать, что для кодирования каждого из цветов достаточно одного бита, при этом нуль в бите означает, что в суммарном цвете данный основной отсутствует, а единица — присутствует. Следовательно, для кодирования одного цветного пиксела потребуется три бита — по одному на каждый цвет. Пусть первый бит соответствует красному цвету, второй зеленому, а третий — синему. Тогда код 101_2 обозначает сиреневый цвет (красный есть, зеленого нет, синий есть), а код 110_2 — желтый цвет (красный есть, зеленый есть, синего нет). При такой схеме кодирования каждый пиксел может быть окрашен в один из восьми возможных цветов. Если каждый из цветов кодировать с помощью одного байта, как это принято для реалистического монохромного изображения, то появится возможность передавать по 256 оттенков каждого из основных цветов. Всего в этом случае обеспечивается передача 256 × 256 × 256 = 16 777 216 различных цветов, что довольно близко к реальной чувствительности человеческого глаза. Таким образом, при данной схеме кодирования цвета на изображение одного пиксела требуется три байта, или 24 бита памяти. Этот способ представления цветной графики принято называть режимом **True Color** (true color — точный цвет), или **полноцветным** режимом. Отметим, что полноцветный режим требует очень много памяти. Так, для обсуждавшегося ранее растра 800 × 600 при использовании метода RGB требуется 1 440 000 байт ≈ 1,37 Мбайт. В целях экономии памяти разрабатываются различные режимы и графические форматы, которые немного хуже передают цвет или само изображение, но требуют гораздо меньше памяти.

Большинство действий, которые приходится выполнять над графическими данными в пиксельном формате, сводятся к замене текущего цвета пиксела другим цветом. Например, стирание какого-либо участка изображения — это не что иное как замена цветов всех пикселов стираемого участка цветом фона рисунка. Следовательно, и для графических данных элементарные действия, которые должен «уметь» выполнять компьютер, сводятся к сравнению двух двоичных кодов и замене одного кода другим.

2.3. Числовые данные

Вообще говоря, запись любого числа может рассматриваться как часть текста, например: «В 1812 году Наполеон напал на Россию». В данном случае последовательность символов «1812» является частью текста, и знаки «1», «2» и «8», — это обычные символы алфавита, для которых существенными являются только действия сравнения и замены одного символа другим. Однако в задачах обработки информации очень часто приходится выполнять разнообразные математические операции над числами, например сложение, вычитание, умножение и т. д. Учитывая это принципиальное отличие, данные, над которыми должны выполняться математические операции, выделяют в особую группу числовых данных, и для них применяются специальные способы кодирования. Обсудим эти способы, предполагая, что читатель знаком с основными понятиями, касающимися двоичных и шестнадцатеричных систем счисления.

2.3.1. Форматы представления чисел в компьютере

Напомним, что исчерпывающе полный набор правил кодирования той или иной разновидности информации в компьютере принято называть *форматом* данных. Для представления числовых данных в компьютерах используются два принципиально разных формата: формат **с фиксированной точкой** (запятой) и формат **с плавающей точкой** (запятой).

В названиях форматов речь идет о знаке, с помощью которого целая часть числа отделяется от его дробной части. В обычной практике записи чисел для этого используется запятая, а в программировании целая часть числа отделяется от дробной точкой. Поэтому в литературе в зависимости от предпочтений авторов используется как термин «фиксированная точка», так и термин «фиксированная запятая». Это же относится и ко второму формату.

Формат с фиксированной точкой предназначен для *абсолютно точного* представления *целых* чисел. В программировании эти числа относятся *к целому типу*, в то время как формат с плавающей точкой используется для представления только нецелых, *приближенных* чисел. В программировании такие числа относятся к *вещественному* типу. Напомним, что вещественные числа возникают в задачах в результате различных измерений (например, измерений веса тела или его длины), которые, как известно, всегда выполняются с некоторой погрешностью, приближенно.

Как выяснится немного позже, возможности одного байта для кодирования чисел довольно малы, поэтому числа обычно занимают несколько соседних байтов, то есть поле, длина которого зависит от используемого формата.

2.3.2. Форматы целых чисел

Существуют две модификации формата с фиксированной точкой, которые принято называть его **беззнаковым** и **знаковым представлениями**. Беззнаковое пред-

ставление формата используется для работы с целыми *неотрицательными* числами, а существующее в нескольких вариантах знаковое — для работы как с положительными, так и с отрицательными целыми числами.

Беззнаковое представление формата с фиксированной точкой

В беззнаковом представлении целого числа используется *прямой двоичный код*, который представляет собой запись этого числа в двоичной системе счисления. При этом все разряды занятого числом поля содержат его значащие цифры. Точка, отделяющая целую часть числа от дробной, считается расположенной, *фиксированной справа от крайнего правого разряда*. Следовательно, под дробную часть числа отводится *нулевое* количество разрядов, и в данном варианте кодировки возможна работа только с целыми числами. Постоянное расположение, фиксация позиции точки дала название формату — с фиксированной точкой.

Пусть N — длина используемого поля в битах, тогда в нем может быть записано N-разрядное двоичное число, и следовательно, в этом поле могут быть представлены коды целых чисел x, максимальное из которых равно $2^N - 1$. Пусть далее $Z_N^0 = \{x \in \mathbb{Z} | 0 \le x \le 2^N - 1\}$, где $\mathbb{Z}$ — *математическое* множество целых чисел. Тогда формально можно записать, что беззнаковым кодом могут быть представлены только числа $x \in Z_N^0$.

Для кодирования чисел в формате с фиксированной точкой используются поля длиной 1, 2 или 4 байта, поэтому N может быть равно 8, 16 или 32. В табл. 2.3 приведены обычно используемые в программировании названия соответствующих этим полям целых типов и диапазоны их возможных значений.

Таблица 2.3. Диапазоны представления беззнаковых целых чисел

Название	Длина, байт	Диапазон чисел (от 0 до $2^N - 1$)	
byte, unsigned char	1	$0...2^8 - 1$	0...255
word, unsigned int	2	$0...2^{16} - 1$	0...65 535
unsigned long	4	$0...2^{32} - 1$	0...4 294 967 295

На практике иногда возникают задачи определения машинного кода заданного числа, а также определения числа по его коду. При использовании беззнакового представления формата с фиксированной точкой эти задачи решаются довольно просто.

Получение беззнакового кода целого числа. Чтобы получить машинный код целого неотрицательного числа в беззнаковом представлении формата с фиксированной точкой, следует перевести заданное число в двоичную систему счисления и записать полученный код в выделенное для него поле.

Пусть, например, задано число 77_{10}. Переводя его в двоичную систему счисления, получим 1001101_2. Искомый код в однобайтном поле имеет вид $0100\ 1101_2$, в двухбайтном — $0000\ 0000\ 0100\ 1101_2$, а в четырехбайтном — $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0100\ 1101_2$.

Видно, что запись машинных кодов непосредственно в двоичном виде очень громоздкая. Гораздо компактнее выглядят те же самые машинные коды, но в шестнадцатеричном виде: $4D_{16}$, 00 $4D_{16}$ и 00 00 00 $4D_{16}$. Поэтому шестнадцатеричное представление используется для отображения содержимого полей памяти гораздо чаще, чем двоичное. При этом двоичные коды группируют по четыре цифры, а шестнадцатеричные — по две. Таким образом, содержимое одного байта изображается *двумя группами из четырех двоичных цифр* или *одной группой из двух шестнадцатеричных цифр*.

Отметим, что выбор длины поля, которое отводится под запись кода числа, зависит от ряда обстоятельств, и не всегда возможно выбрать поле любой допустимой длины, как в рассмотренном примере. При выборе длины поля самым важным фактором является значение числа, код которого должен быть записан в это поле. Из табл. 2.3 следует, что, например, для числа 155_{10}, имеющего код $9B_{16}$, при использовании обсуждаемого способа можно выбрать поле любой возможной длины. А, скажем, код числа 497_{10} в однобайтовое поле не поместится, так как оно больше, чем максимально допустимое для одного байта число 255_{10}. Число 497_{10} имеет беззнаковый код $1F1_{16}$, и для его записи могут быть использованы только двух- или четырехбайтовые поля. А если рассматривается число, большее чем 65 535_{10}, например 567 398 104_{10}, то для его кода 21 D1 CE $D8_{16}$ потребуется уже четырехбайтовое поле. Из этой таблицы также следует, что числа большие, чем 4 294 967 295_{10}, вообще не могут быть закодированы таким способом.

Определение значения числа по его беззнаковому коду. Чтобы определить значение числа, код которого задан в беззнаковом представлении формата с фиксированной точкой, достаточно, считая весь этот код двоичным (или шестнадцатеричным) числом, перевести его в десятичную систему счисления.

Пусть, например, задан занимающий два байта памяти машинный код $010B_{16}$ некоторого числа в обсуждаемом формате. Переходя в десятичную систему счисления, получим, что это код числа 267_{10}.

Знаковые представления формата с фиксированной точкой

В истории развития архитектуры компьютеров использовались четыре различных варианта представления знаковых чисел:

- система со знаком;
- обратный код, поразрядное дополнение или код с дополнением до единицы;
- дополнительный, комплементарный код или код с дополнением до двух;
- система со смещением.

В настоящее время первые две системы устарели и практически вышли из употребления. Тем не менее по ходу изложения материала мы затронем и эти устаревшие системы.

Очевидно, что для машинного представления знаковых чисел нужно определенным образом закодировать знак числа и его *модуль*. Так как существует всего два знака чисел, «+» и «–», то самое простое напрашивающееся решение состоит

в том, чтобы представить код знака одной двоичной цифрой и выделить под него один из разрядов поля, а во всех остальных разрядах записывать *модуль* числа, кодируя его, например, с помощью *прямого двоичного кода*, так же как кодируются беззнаковые целые числа. Такой способ называется **системой кодирования со знаком**. Код знака числа принято размещать в *самом левом* разряде поля, который в связи с этим принято называть **знаковым битом**. По традиции знак «плюс» кодируется нулем, а знак «минус» — единицей. Таким образом, если в знаковом бите находится *нуль*, это означает, что остальные биты поля содержат модуль *положительного* числа, а если знаковый бит занят *единицей*, то в них находится модуль *отрицательного* числа. Заметим, что можно было бы договориться и о другом способе кодирования знака, но по ряду причин выбран именно этот способ.

На рис. 2.4 показано размещение знаковых битов в полях разной длины. В однобайтовом поле разряды с 0-го по 6-й содержат код модуля числа, а код знака занимает 7-й разряд. В двухбайтовом поле код модуля занимает разряды с 0-го по 14-й, код знака — 15-й разряд. В четырехбайтовом поле код модуля расположен в разрядах с 0-го по 30-й, а код знака находится в 31-м разряде.

Рис. 2.4. Разрядная сетка системы кодирования со знаком

Обратите внимание на то, что в знаковом представлении различные биты кода играют различную роль, в то время как во всех рассматривавшихся ранее вариантах кодирования все цифры кода играют одну и ту же роль, например образуют код отдельного символа текста или являются значащими цифрами беззнакового кода целого числа. Закрепление за разрядами поля конкретных функций хранения различных элементов кода принято называть **разрядной сеткой**.

Итак, в рассматриваемом варианте кодирования для представления *модулей* целых чисел предлагается использовать *прямой двоичный код числа*. Возьмем, например, числа $+4_{10}$ и -4_{10} с прямым двоичным кодом модуля 100_2. Тогда однобайтовый код числа $+4_{10}$ есть $0|000\ 0100_2$, или 04_{16}, а такой же код числа -4_{10} — это $1|000\ 0100_2$, или 84_{16}. Еще пример: число $+127_{10}$ имеет в этой системе код $0|111\ 1111_2$ ($7F_{16}$), а число -127_{10} — код $1|111\ 1111_2$ (FF_{16}). В приведенных двоичных кодах для наглядности знаковый бит отделен от битов модуля вертикальной чертой, которая, естественно, в реальных машинных кодах отсутствует.

Поскольку один из N битов поля отводится под код знака, под запись кода модуля остается $N - 1$ битов. Следовательно, в таком поле могут быть закодированы

целые числа, модули которых удовлетворяют условию $|x| \le 2^{N-1} - 1$. Пусть $Z_N^1 = \{x \in \mathbb{Z} \mid 1 - 2^{N-1} \le x \le 2^{N-1} - 1\}$. В системе кодирования со знаком могут быть представлены только числа $x \in Z_N^1$.

Посмотрим теперь, к каким последствиям приведет принятие предлагаемого варианта кодирования знаковых целых чисел. Во-первых, оказывается, что числу 0_{10} соответствуют два различных кода: код $0000\ 0000_2$ ($+0_{10}$) и код $1000\ 0000_2$ (-0_{10}). Такая неоднозначность крайне нежелательна, так как необходимо либо дополнительно учитывать ее при различных проверках аппаратными средствами компьютера, либо предусматривать отдельные проверки в программах. Во-вторых, и это самое главное, возникает специальная арифметика с совершенно непривычными правилами выполнения самых обычных действий. Так, по правилам обычной арифметики, сложение чисел $+4_{10}$ и -4_{10} дает в результате 0_{10}. А теперь выполним сложение для полученных ранее кодов этих чисел: $00000100_2 + 10000100_2 = 10001000_2$. Как видим, получен совершенно неожиданный результат: вместо ожидавшегося кода числа 0_{10} сложение дало код числа -8_{10}. Этот результат является следствием неудачного выбора способа кодирования. Его нужно выбирать исходя из *логики использования кода*, а не из «напрашивающегося» или «очевидного» на первый взгляд подхода. Основное требование при выборе системы кодирования чисел состоит в том, что полученный код должен удовлетворять правилам выполнения сложения и вычитания в двоичной системе счисления. В связи с этим код каждого следующего *положительного* числа должен получаться *прибавлением* единицы к коду текущего числа, а код каждого следующего *отрицательного* числа должен получаться *вычитанием* единицы из кода текущего числа. Построенный таким образом код принято называть **дополнительным**, **с дополнением до двух** или **комплементарным** (от complementary — дополняющий).

Для простоты рассуждений рассмотрим построение этого кода на примере четырех битов. Подчеркнем, что при построении кодов знаковых чисел знаковый бит ни при каких условиях не может быть занят значащими цифрами кода модуля, в него не должен выполняться перенос и в нем нельзя производить заем. Итак, код числа 0_{10} — это 0000_2. Код следующего положительного числа $+1_{10}$ получаем прибавлением к нему единицы: 0001_2. Аналогичным образом получаем код числа $+2_{10}$: 0010_2. Дальнейший процесс получения кодов положительных чисел изображен на рис. 2.5, *слева*. Отметим, что при записи кода в четыре бита этот процесс можно продолжить только до числа $+7_{10}$, имеющего код 0111_2. Переход к следующему положительному числу $+8_{10}$ с кодом 1000_2 приведет к занятию знакового бита старшей цифрой кода модуля. Следовательно, числа, большие $+7_{10}$, таким способом изобразить нельзя.

Рассмотрим теперь получение кодов отрицательных чисел. Для получения кода числа -1_{10} вычтем из кода числа 0_{10} единицу: $0000_2 - 1_2$. Это возможно только если условиться о том, что необходимый для такого вычитания заем выполняется из воображаемого дополнительного разряда: $10000_2 - 1_2 = 1111_2$. Дальнейшее получение кодов отрицательных чисел не вызывает затруднений и показано на рис. 2.5, *справа*. Например, код числа -2_{10} получается так: $1111_2 - 1_2 = 1110_2$. Очевидно, что получение кодов чисел, меньших, чем -8_{10}, в случае использования для кода всего четырех битов невозможно, так как, например, для получения кода

числа -9_{10} нужно выполнить заем из знакового бита: $1000_2 - 1_2 = 0111_2$. К тому же, получающийся при этом код 0111_2 уже используется как код числа $+7_{10}$.

	$0 \rightarrow 0000_2$
$+7 \rightarrow 0111_2$	$-1 \rightarrow 1111_2$
$+6 \rightarrow 0110_2$	$-2 \rightarrow 1110_2$
$+5 \rightarrow 0101_2$	$-3 \rightarrow 1101_2$
$+4 \rightarrow 0100_2$	$-4 \rightarrow 1100_2$
$+3 \rightarrow 0011_2$	$-5 \rightarrow 1011_2$
$+2 \rightarrow 0010_2$	$-6 \rightarrow 1010_2$
$+1 \rightarrow 0001_2$	$-7 \rightarrow 1001_2$
$0 \rightarrow 0000_2$	$-8 \rightarrow 1000_2$

Рис. 2.5. Получение дополнительных кодов

Отметим, что название «дополнительный» код возникло потому, что в качестве кода отрицательного числа фактически выбирается D — дополнение n-разрядного прямого кода модуля P этого числа до числа $M = 2n$: $D = M - P$. Вновь возьмем $n = 4$ и найдем код числа -4. В данном случае $M = 10000_2$, прямой четырехразрядный код модуля обсуждаемого числа $P = 0100_2$, и, следовательно, его дополнение до M равно $D = 10000_2 - 0100_2 = 1100_2$, что совпадает с полученным прямыми рассуждениями дополнительным кодом.

Способ кодирования знаковых чисел, основанный на использовании *дополнительного* кода, устраняет все отмеченные ранее недостатки применения системы кодирования со знаком. Во-первых, необходимые арифметические свойства удовлетворяются автоматически по способу построения кода. Например, при сложении кодов 0100_2 ($+4_{10}$) и 1100_2 (-4_{10}) получается код 10000_2, старшая единица которого не помещается в используемые четыре разряда и *отбрасывается*. Таким образом, остается код 0000_2, который в точности соответствует нужному результату. Отметим, что это общий технический прием при выполнении действий со знаковыми числами. Он аналогичен приему, использованному в рассуждениях при получении дополнительного кода числа -1. Можно считать, что во время выполнения аналогичных операций возвращается заем, выполненный ранее из *воображаемого дополнительного разряда*.

Во-вторых, исчезла неоднозначность кодирования нуля. В самом деле, код 1000_2, который раньше вместе с кодом 0000_2 использовался для кодирования нуля, оказался закрепленным за кодом максимального по модулю отрицательного числа -8_{10}. Это привело к тому, что диапазоны представления положительных и отрицательных чисел стали несимметричными. В самом деле, если для записи кода используется только 4 бита, то при выборе *любого* способа кодирования возможно формирование всего $2^4 = 16$ различных кодов. Из них при применении системы со знаком 14 кодов закреплено за ненулевыми числами из симметричного диапазона от -7_{10} до $+7_{10}$, и еще два кода, 0000_2 и 1000_2, соответствуют нулю. Применение дополнительного кода позволяет изобразить шестнадцать целых чисел (вместе с нулем) из несимметричного диапазона от -8_{10} до $+7_{10}$. При этом под нуль занят всего один код 0000_2, а закрепление второго кода 1000_2 за числом -8_{10} как раз и приводит к появлению несимметричности диапазона.

Приведенные рассуждения не зависят от длины поля. Поэтому при использовании дополнительных кодов для полей длиной N бит в соотношении $1-2^{N-1} \le x \le 2^{N-1}-1$, описывающем диапазон кодируемых чисел в системе со знаком, левую границу нужно сместить на единицу в *отрицательную* сторону. Следовательно, при использовании *дополнительных* кодов в поле длиной N бит можно закодировать числа $x \in Z_N^2$, где $Z_N^2 = \{x \in \mathbb{Z} | -2^{N-1} \le x \le 2^{N-1}-1\}$ (табл. 2.4).

Таблица 2.4. Диапазоны представления знаковых целых чисел

Название	Длина, байт	Диапазон (от -2^{N-1} до $2^{N-1}-1$)	
Shortint	1	$-2^7 \ldots 2^7 - 1$	−128 ... 127
Integer	2	$-2^{15} \ldots 2^{15} - 1$	−32 768 ... 32 767
Longint	4	$-2^{31} \ldots 2^{31} - 1$	−2 147 483 648 ... 2 147 483 647

ВНИМАНИЕ

В дальнейшем для простоты изложения представление целых чисел в дополнительном коде формата с фиксированной точкой называется просто знаковым представлением (кодированием) чисел.

При сравнении диапазонов чисел из табл. 2.3 и 2.4 для беззнакового и знакового представлений может показаться, что диапазон изображаемых чисел в случае знакового представления уже, чем в случае беззнакового варианта. Однако это не так, поскольку общее количество различных кодов зависит только от длины используемого поля и никак не связано с используемым форматом.

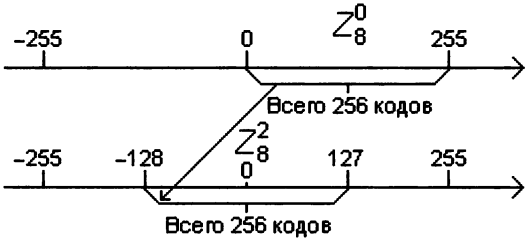


Рис. 2.6. Сравнение диапазонов беззнакового и знакового кодирования в однобайтном поле

На рис. 2.6 сравниваются множества Z_8^0 и Z_8^2 кодируемых в одном байте чисел для беззнакового (*вверху*) и знакового (*внизу*) представлений. Для беззнакового представления при $N = 8$ битов возможно кодирование множества целых из интервала от 0 до 255 (табл. 2.3), а для знакового представления — от −128 до +127 (табл. 2.4). Общее количество кодов и в первом, и во втором случае равно 256. Просто эти коды оказалась закрепленными за целыми числами из различных диапазонов. Фактически, для знакового кодирования весь набор кодов однобайтового поля сдвинут вдоль числовой оси как единое целое назад на 128 единиц.

Способы получения дополнительных кодов целых чисел. Понятно, что для практического применения рассмотренный способ получения дополнительного

кода малопригоден. Поэтому рассмотрим два простых способа получения дополнительного кода *по известному прямому коду*. Первый способ можно применять, когда используется двоичная система счисления.

1. Прямой код записывается в выбранное поле длиной 1, 2 или 4 байта.
2. Прямой код инвертируется (обращается), то есть каждая цифра 0 кода заменяется цифрой 1, и наоборот, каждая цифра 1 кода заменяется цифрой 0. Заметим, что полученный таким образом код называется **обратным, с дополнением до единицы** или **с поразрядным дополнением**. К обратному коду прибавляется единица.

Пусть, например, имеется прямой код $100\ 1101_2$ и выбрано однобайтовое поле. Тогда исходным будет код в виде $0100\ 1101_2$, его обратный код — $1011\ 0010_2$, а дополнительный — $1011\ 0011_2$.

Важным моментом в получении правильного конечного результата является первоначальная запись кода в поле подходящей длины. Возьмем, например, прямой код 100_2. Его инвертирование дает обратный код 011_2, а последующее добавление единицы — дополнительный 100_2. Совпадение прямого и дополнительного кодов свидетельствует об ошибке, так как в данном случае однозначность кодирования является одним из основных предъявляемых к коду требований. Ошибка состоит в том, что сделана попытка работать с несоответствующим значению числа количеством битов. В самом деле, прямой код 100_2 числа 4 занимает $N = 3$ бита. Но, как следует из вышеизложенного, обсуждаемым способом в этом количестве битов можно представить коды чисел только из диапазона от −4 до +3. Для получения правильного результата необходимо взять хотя бы 4 бита. Тогда получится правильный результат: прямой код 0100_2, обратный — 1011_2, а дополнительный — 1100_2. Итак, при выборе длины поля для записи кода числа необходимо учитывать приведенные в табл. 2.4 соответствия между длиной поля и диапазоном представимых в нем чисел.

Отметим, что если используется прямой код числа, модуль которого принадлежит допустимому для данного поля диапазону, то в знаковом (крайнем слева) бите *обязательно* находится 0 (этого не было у кода 100_2!). Тогда переход к дополнительному коду автоматически приведет к формированию в знаковом бите нужной единицы.

Второй способ удобно применять и для двоичных, и для шестнадцатеричных кодов. Для получения дополнительного кода нужно прямой код вычесть из числа $M = 2^8N$ для двоичной системы или 16^2N для шестнадцатеричной, где N — длина используемого поля в байтах. Так, для однобайтовых ($N = 1$) полей и двоичного кода вычитать прямой код следует из числа $2^8 = 1\ 0000\ 0000_2$. А для того же поля и шестнадцатеричного кода вычитание должно выполняться из числа $16^2 = 100_{16}$. Пусть, например, прямой код равен $100\ 1101_2$, его вычитание из $1\ 0000\ 0000_2$ дает в результате $1011\ 0011_2$. Очевидно, что, работая в шестнадцатеричной системе счисления, можно получить тот же результат, но в значительно более компактном виде: вычитая прямой код $4D_{16}$ из 100_{16}, получаем тот же дополнительный код в виде $B3_{16}$.

Получение знакового кода заданного числа. Чтобы получить машинный код целого числа в знаковом представлении формата с фиксированной точкой, следует:

1) перевести модуль этого числа в двоичную систему счисления;
2) по табл. 2.4 выбрать поле, длина которого обеспечивает возможность записи знакового кода рассматриваемого числа;
3) если число положительное, записать полученный код в выбранное поле.
4) если число отрицательное, получить в выбранном поле его дополнительный код.

Пусть, например, задано число $+77_{10}$. Перевод модуля этого числа в двоичную систему счисления дает $77_{10} \rightarrow 1001101_2$. Так как рассматривается положительное число, то осталось записать его код в выбранное поле. По табл. 2.4 находим, что для этого числа может быть использовано поле любой длины. Отсюда для однобайтового поля получаем: $0100\ 1101_2$, для двухбайтового — $0000\ 0000\ 0100\ 1101_2$, и для четырехбайтового — $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0100\ 1101_2$.

Попутно заметим, что, если число $x \geq 0$ принадлежит множеству Z_N^2, то оно принадлежит и множеству Z_N^0, то есть и знаковый, и беззнаковый коды этого числа могут быть записаны в поле выбранной длины N. В этом случае его *знаковый* и *беззнаковый* коды в выбранном поле *совпадают*. Это видно хотя бы на примере рассматривавшихся ранее кодов числа $+77_{10}$. А теперь возьмем, скажем, $x = +155_{10}$, прямой код модуля которого $1001\ 1011_2$ или $9B_{16}$. Ясно, что для однобайтового поля $x \in Z_8^0$, но, с другой стороны, $x \notin Z_8^2$. Это значит, что беззнаковый код числа может быть записан в однобайтовое поле, а знаковый — нет. То, что код 10011011_2 не может быть знаковым однобайтовым кодом положительного числа, видно и по занятому единицей знаковому биту. В этом случае, как известно, в поле находится код отрицательного числа. Как будет показано далее, знаковый код $9B_{16}$ соответствует числу -101_{10}, в то время как знаковые коды $009B_{16}$ и $00009B_{16}$ представляют число $+155_{10}$. С другой стороны все коды $9B_{16}$, $009B_{16}$ и $00009B_{16}$, рассматриваемые как беззнаковые, соответствуют числу $+155_{10}$. Аналогичная ситуация наблюдается для всех неотрицательных чисел, которые для поля выбранной длины N принадлежат Z_N^0 и не принадлежат Z_N^2: $\left(x \in Z_N^0\right) \wedge \left(x \notin Z_N^2\right)$.

Теперь получим знаковое представление числа -77. Прямой код модуля этого числа уже найден, это $1001\ 101_2$. Переходя любым из способов к дополнительному коду, для однобайтового поля получим $1011\ 0011_2$ или $B3_{16}$. В случае использования двух- и четырехбайтовых полей работать с двоичными кодами неудобно. Поэтому перейдем к шестнадцатеричным кодам. Итак, для однобайтового поля прямой код модуля числа -77 равен $4D_{16}$, и тогда искомый дополнительный код есть $100_{16} - 4B_{16} = B3_{16}$. Для двухбайтового поля находим: $10000_{16} - 4D_{16} = FFB3_{16}$, и, наконец, для четырехбайтового — $100000000_{16} - 4D_{16} = FFFFFFB3_{16}$.

В связи с тем, что для записи кодов знаковых чисел могут быть использованы поля различной длины, на практике возникает задача: по имеющемуся коду числа в поле некоторой длины найти его код в поле большей или меньшей длины. На рис. 2.7 показаны знаковые представления для чисел $+77$ и -77 в полях разной длины. В двоичных кодах вертикальной чертой отделен знаковый бит поля. Сравнение этих результатов показывает, что при переходе к полю большей длины

свободные слева биты поля заполняются кодом знакового бита 0_2 (0_{16}) для положительных и 1_2 (F_{16}) — для отрицательных чисел. Это правило называется *правилом размножения знака*. Переход к полям меньшей длины в тех случаях, когда он возможен, осуществляется отбрасыванием двух, четырех или шести цифр 0_{16} или F_{16} или же соответствующего количества тетрад 0000_2 или 1111_2.

Коды числа $+77_{10}$ в стандартных полях

Поле 1 байт	$0\|100\ 1101_2$ или	$4D_{16}$
Поле 2 байта	$0\|000\ 0000\ 0100\ 1101_2$ или	$00\ 4D_{16}$
Поле 4 байта	$0\|000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0100\ 1101_2$ или	$00\ 00\ 00\ 4D_{16}$

Коды числа -77_{10} в стандартных полях

Поле 1 байт	$1\|011\ 0011_2$ или	$B3_{16}$
Поле 2 байта	$1\|111\ 1111\ 1011\ 0011_2$ или	$FF\ B3_{16}$
Поле 4 байта	$1\|111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1011\ 0011_2$ или	$FF\ FF\ FF\ B3_{16}$

Рис. 2.7. Размножение знака при переходе к полям большей длины

Определение значения числа по его знаковому коду. Чтобы определить значение числа, для которого задан его машинный код в знаковом представлении формата с фиксированной точкой, нужно сделать следующее.

1. По знаковому биту определить знак числа. Для отрицательных чисел в двоичной кодировке крайняя слева цифра равна 1_2, а в шестнадцатеричной эта цифра должна быть больше 7_{16}. В противном случае поле содержит прямой код положительного числа.
2. Если число отрицательное, то в поле находится дополнительный код, от которого по формуле $P = M - D$ следует перейти к прямому коду.
3. Перевести число из двоичной или шестнадцатеричной системы счисления в десятичную.
4. Приписать результату соответствующий знак числа.

Пусть, например, задан код $005A_{16} = 0000\ 0000\ 0101\ 1010_2$ и известно, что это знаковое представление формата с фиксированной точкой. По первой цифре шестнадцатеричного кода 0_{16} или знаковому биту 0_2 определяем, что в поле находится код положительного числа, перевод которого в десятичную систему счисления дает $+90_{10}$. Еще пример. Пусть в том же формате задан код $FFE8_{16}$ или $1111\ 1111\ 1110\ 1000_2$. Первая цифра кода $F_{16} \geq 8_{16}$ или равный 1_2 знаковый бит являются признаками отрицательного числа. Переход к прямому коду дает $10000_{16} - FFE8_{16} = 18_{16}$ или 24_{10}. Окончательно находим, что рассматриваемый код $FFE8_{16}$ соответствует числу -24_{10}. Если же код $FFE8_{16}$ рассматривать как беззнаковое представление числа, то он соответствует числу $65\ 512_{10}$.

ВНИМАНИЕ

Один и тот же код в разных представлениях и, тем более, разных форматах может соответствовать различным числам.

В заключение кратко обсудим систему кодирования **со смещением**, которая в англоязычной литературе называется **системой excess 2N – 1**. Эта система кодирования характеризуется целым положительным числом — константой смещения K, которое прибавляется к любому кодируемому числу x с тем, чтобы сумма исходного числа и константы смещения $x + K$ попала в диапазон допустимых для поля выбранной длины N беззнаковых чисел Z_N^0: $x + K \in Z_N^0$. В качестве *кода исходного числа* выбирается *беззнаковый код суммы*. Таким образом, множество кодируемых чисел $\overline{Z}_N^K$ в системе с константой смещения K есть $\overline{Z}_N^K = \{x \notin Z | -K \le x \le 2^N - K - 1\}$. Пусть, например, $N = 8$ и $K = 64$. Тогда $\overline{Z}_N^{64} = \{x \in Z | -64 \le x \le 191\}$, а код, например, числа $x = -4_{10}$ определяется как беззнаковый код числа 60_{10}, равный $0011\ 1100_2$ или $3C_{16}$. В определенном смысле эта операция является обратной по отношению к изображенной на рис. 2.7: диапазон, включающий числа разного знака, с помощью константы K смещается как единое целое вдоль числовой оси в конечное положение, совпадающее с диапазоном беззнакового кодирования.

Если взять специальный случай, когда константа $K = 2N$, где N — длина поля в битах, то окажется, что система с таким смещением весьма близка к дополнительному коду. Возьмем, например $N = 8$, тогда $K = 128$ и $\overline{Z}_N^{128} = \{x \in Z | -128 \le x \le 127\}$. Как можно заметить, в данном случае диапазоны системы со смещением и дополнительного кода *совпадают*. Практически одинаковы и получаемые при этом коды чисел. Например, в качестве кода числа –4 в этой системе выбирается беззнаковый код числа 124_{10}, равный $0111\ 1100_2$ или $7C_{16}$. Если сравнить полученный результат с *дополнительным* кодом $1111\ 1100_2$ того же самого числа –4, то легко увидеть, что они различаются только в знаковом бите. Другими словами, код числа в системе со смещением с константой $K = 2N$ совпадет с *дополнительным* кодом, у которого инвертирован знаковый бит.

Как мы увидим в дальнейшем, в настоящее время система кодирования со смещением используется как один из элементов представления чисел в формате с плавающей точкой.

Ранее отмечалось, что устройство, осуществляющее арифметические операции над числами в двоичной системе счисления, должно обеспечивать выполнение действий сложения, вычитания и сдвига, так как умножение и деление сводятся именно к этим операциям. В связи с тем, что вычитание всегда можно заменить сложением с отрицательным числом, становится понятно, что процессоры компьютеров могут «обойтись» и без операции вычитания. Для обработки целочисленных данных им достаточно «уметь» выполнять сложение и сдвиги для двоичных кодов.

2.3.3. Формат вещественных чисел

Во многих расчетных задачах используются величины, которые являются результатами всевозможных измерений или получены с помощью различных математических операций над измеренными значениями. Их отличительной особенностью является принципиально приближенный характер, их значения *никогда* не бывают в точности равными целому числу. Такого рода величины и их значения принято называть **вещественными**.

Не следует путать понятия вещественного числа в информатике и действительного числа в математике. Математическое множество *действительных* чисел $\mathbb{R}$ содержит в себе множество целых чисел $\mathbb{Z}$: $\mathbb{Z} \subset \mathbb{R}$. Множество *вещественных* чисел информатики $\mathbb{R}^*$ образуется совершенно по другому принципу: значение числа обязательно должно быть приближенным. Поэтому множество вещественных данных не содержит целых чисел, которые по своей природе являются точными: $\mathbb{Z}_N^0 \notin R^*$ и $\mathbb{Z}_N^0 \notin R\hat{E}^*$.

Обычно используемая запись числа в виде $\pm a, b$, содержащем целую (a) и дробную (b) части, считается основной формой записи вещественных чисел. В естественных науках довольно часто вместо основной формы числа используется его запись в виде произведения двух сомножителей, один из которых является основанием системы счисления в некоторой степени, например: $2{,}9 \cdot 10^{18}$ или $0{,}091 \cdot 10^{-30}$. Чаще всего такая запись используется во время работы с очень большими или очень маленькими по модулю числами. При этом достигается значительная экономия во времени, в наглядности, в простоте восприятия содержащего такие числа текста. Сравните, например, способы записи одного и того же числа: 2 900 000 000 000 000 000 и $2{,}9 \cdot 10^{18}$, а также 0,00000000000000000000000000000091 и $0{,}091 \cdot 10^{-30}$.

Запись вида $\pm m \times p^{\pm q}$ называется **формой с порядком** или **экспоненциальной формой** вещественного числа. Некоторые авторы используют также название **полулогарифмическая форма**. Сомножитель m принято называть **мантиссой**, а степень $\pm q$, в которую возводится основание p системы счисления, — **порядком** числа. Так, для числа $2{,}9 \cdot 10^{18}$ мантисса $m = 2{,}9$, основание $p = 10$, а порядок $q = 18$.

Следует обратить внимание на то, что приведенная ранее запись числа $2{,}9 \cdot 10^{18}$ в основной форме формально выглядит как целое число, то есть как точное значение, и следовательно, относить это число к вещественным вроде бы нельзя. В таких случаях нужно принимать во внимание исходный принцип: если значение получено в результате измерения или расчета, выполненного с привлечением приближенных величин, то имеется вещественное число, которое в результате округления приближенного значения в основной форме выглядит как целое.

Нормализованные числа

К сожалению, обсуждаемая форма записи чисел с точки зрения ее использования для представления данных в компьютерах обладает существенным недостатком: она *неоднозначна*. Изменяя положение запятой, отделяющей целую часть числа от дробной, и соответствующим образом увеличивая или уменьшая порядок, можно получить сколь угодно много возможных вариантов записи равных по значению чисел, например:

$$18{,}456 = 1{,}8456 \cdot 10^{+1} = 0{,}18456 \cdot 10^{+2} = 184{,}56 \cdot 10^{-1} = 1845{,}6 \cdot 10^{-2}.$$

Чтобы избавиться от указанной неоднозначности, на значение мантиссы m накладывают какое-либо ограничение, например, $0{,}1 \le m < 1$ или $1 \le m < 10$. Отметим, что в стандартах последнего времени закрепилось использование ограничения

$1 \le m < 10$, которое в случае двоичной системы счисления означает, что *целая* часть мантиссы *всегда* должна быть равна единице.

Внешний вид обсуждаемых ограничений *не зависит* от используемой системы счисления. Этот вывод следует из очевидного соображения: в любой системе счисления числа 0,1, 1 и 10 означают основание системы p в минус первой, нулевой и в первой степени соответственно. Следовательно, соотношения $p^{-1} \le m < p^0$ и $p^0 \le m < p^1$ могут быть записаны в приведенном ранее виде в любой системе счисления. В таких случаях говорят, что форма записи **инвариантна** (от фр. invariant — неизменяющийся) по отношению к некоторому фактору (в данном случае к системе счисления).

ВНИМАНИЕ

Вещественные числа, записанные в форме $\pm m \times p^{\pm q}$, где $1 \le m < 10$, называются нормализованными.

Примеры нормализованных десятичных чисел: $2{,}9 \cdot 10^{18}$, $1{,}8456 \cdot 10^{+1}$, $9{,}1 \cdot 10^{-32}$. Числа $0{,}29 \cdot 10^{19}$, $29{,}0 \cdot 10^{17}$, $0{,}18456 \cdot 10^{+2}$, $18{,}456 \cdot 10^{0}$ и т. д. — не нормализованы.

Выполнение операций над нормализованными числами

При рассмотрении сложения и вычитания над числами в *основной* форме отмечалось, что эти операции выполняются *поразрядно*. Это значит, что в заданной операции участвуют цифры одного и того же разряда, иначе говоря, коэффициенты при одной и той же степени основания системы счисления. По-другому обстоят дела при сложении или вычитании нормализованных чисел. Во-первых, если нормализованные числа имеют различные порядки, то выполнять действие над одноименными разрядами мантиссы нельзя, так как из-за различных значений порядков в одинаковые разряды мантиссы попадают коэффициенты при различных степенях основания системы счисления. Во-вторых, после выполнения действия результат может оказаться ненормализованным.

На рис. 2.8, *а–в* приведены примеры ситуаций, которые могут возникнуть при выполнении арифметических операций над нормализованными числами. На рис. 2.8, *а* показана самая благоприятная ситуация, когда порядки слагаемых одинаковые. В этом случае можно выполнять операцию, как в основной форме — поразрядно. В данном примере результат также оказался нормализованным. На рис. 2.8, *б* слагаемые имеют разные порядки. Поэтому перед выполнением действия с помощью **денормализации** их нужно сделать одинаковыми, выровнять. В данном примере во время денормализации порядок второго слагаемого увеличен на единицу. Соответственно, чтобы число не изменилось, *его мантисса уменьшена в десять раз*. После выполнения действия результат оказался нормализованным. И наконец, на рис. 2.8, *в* приведен пример ситуации, когда не только требуется выполнить денормализацию перед выполнением действия, но и результат оказывается ненормализованным, так что дополнительно требуется выполнять нормализацию. Отметим, что последней ситуации можно избежать, если выполнять денормализацию в сторону большего порядка, как это сделано в примере на рис. 2.8, *б*.

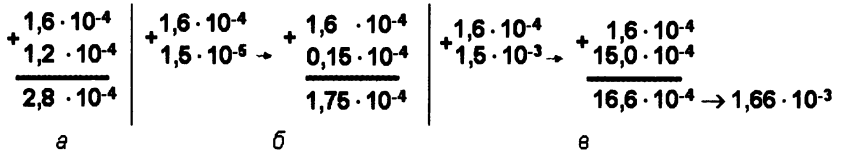


Рис. 2.8. Примеры выполнения операций над нормализованными числами

Обнаруженное усложнение выполнения действий над нормализованными числами является платой за возможность работать с числами из гораздо более широкого диапазона, чем в формате с фиксированной точкой. Заметим также, что используемый для представления вещественных чисел формат с *плавающей* точкой получил свое название в связи с необходимостью *перемещать* во время выполнения арифметических операций текущее положение точки (запятой), отделяющей целую часть числа от дробной.

Общие принципы кодирования чисел в формате с плавающей точкой

Для кодирования вещественных чисел в формате с плавающей точкой используется их нормализованная форма $\pm m \cdot p^{\pm q}$, $1 \le m < 10$. Анализ этой формы показывает, что в системе счисления с *заданным* основанием p код нормализованного числа должен содержать:

1) код знака числа;

2) код нормализованной мантиссы $1 \le m < 10$;

3) код знака порядка;

4) код порядка p.

Для упрощения понимания материала обсуждение основных особенностей этого способа кодировки проведем с привлечением воображаемого устройства, состоящего из восьми десятичных разрядов. В каждый разряд этого устройства, которое мы назовем «учебная ячейка», можно записать *любую* цифру десятичной системы счисления, а не только 0 или 1.

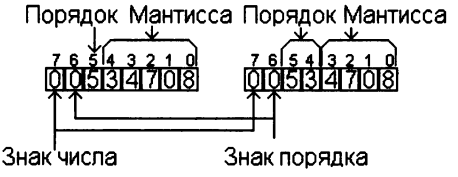


Рис. 2.9. Возможные разрядные сетки учебной ячейки из восьми десятичных разрядов

На рис. 2.9 показаны два из нескольких возможных вариантов распределения разрядов учебной ячейки, которое обеспечивает хранение всех указанных ранее элементов кода нормализованного числа. Более подробно рассмотрим разрядную сетку, показанную на рис. 2.9, *слева*. Крайний слева седьмой разряд ячейки

по традиции отведен под код знака. В следующем, шестом разряде предлагается кодировать знак порядка. Знаки числа и порядка кодируются стандартным образом: 0 соответствует положительному числу, а 1 — отрицательному. Таким образом, в шестом и седьмом разрядах могут находиться только эти две цифры 0 и 1. Пятый разряд отведен под запись порядка, а разряды с нулевого по четвертый содержат значащие цифры *нормализованной* мантиссы.

Если принять предложенную разрядную сетку, то получим, что в изображенной на рис. 2.9 учебной ячейке закодировано число $+3{,}4708 \cdot 10^{+5}$. Разберем подробнее этот переход. Итак, в седьмом разряде находится 0. Его содержимое рассматривается как код знака числа, следовательно, в ячейке находится положительное число. Разряды с нулевого по четвертый содержат значащие цифры 34708 *нормализованной* мантиссы. Нормализованность означает, что ее целая часть содержит ровно одну отличную от нуля цифру. Следовательно, мантисса имеет вид 3,4708. В пятом разряде находится порядок 5, а в шестом — код его знака, 0. Следовательно, порядок числа равен +5. Собирая вместе все результаты, получим, что в ячейке находится код числа $+3{,}4708 \cdot 10^{+5}$.

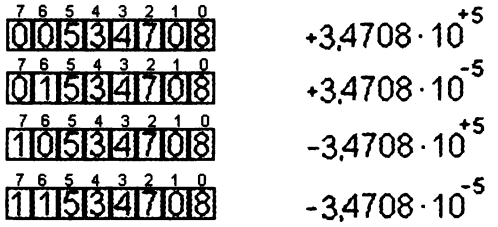


Рис. 2.10. Примеры кодирования чисел в учебной ячейке

На рис. 2.10 приведены примеры получения значения чисел для различных вариантов заполнения знаковых шестого и седьмого разрядов учебной ячейки при одном и том же содержимом остальных ее разрядов.

Теперь можно разобраться с вопросом о множестве вещественных чисел, которые могут быть закодированы таким способом. Для этого следует выяснить, какие максимальное и минимальное *по модулю* числа представимы в учебной ячейке.

Начнем с максимального. Очевидно, что такое число должно иметь самый большой положительный порядок и самую большую мантиссу. В предложенной разрядной сетке под порядок отводится одна десятичная цифра, следовательно, максимально возможный порядок равен +9, а состоящая из пяти цифр нормализованная мантисса равна 9,9999. То есть максимально возможное для данной разрядной сетки число равно $9{,}9999 \cdot 10^{+9}$. Заполняя разряды ячейки соответствующими значениями, получим код максимального числа в виде 00999999. Понятно также, что в учебную ячейку может быть записан код 10999999, соответствующий числу $-9{,}9999 \cdot 10^{+9}$. Итак, мы получили, что числа, модуль которых больше, чем $9{,}9999 \cdot 10^{+9}$, *не могут* быть закодированы обсуждаемым способом.

Теперь рассмотрим возможности учебной ячейки по записи маленьких чисел. Ясно, что *наименьшее* по модулю число должно иметь *минимальный* порядок,

который в данных условиях равен –9, и минимально возможную нормализованную мантиссу, равную 1,0000. Этим требованиям удовлетворяют число $+1{,}0000 \cdot 10^{-9}$ с кодом 01910000 и число $-1{,}0000 \cdot 10^{-9}$ с кодом 11910000. Числа, модули которых меньше, чем $1{,}0000 \cdot 10^{-9}$, данным способом представить *невозможно*. Итак, в обсуждаемой условной ячейке с предложенной разрядной сеткой могут быть представлены числа x, удовлетворяющие условию $1{,}0000 \cdot 10^{-9} \le |x| \le 9{,}9999 \cdot 10^{+9}$. Особо подчеркнем то обстоятельство, что точность всех чисел, которые могут быть закодированы по обсуждаемой схеме, не может быть выше пяти значащих цифр. Это следует из выбранного закрепления за значащими цифрами мантиссы ровно пяти разрядов ячейки. Пусть $\mathbb{R}_5^*$ — множество нормализованных и округленных с точностью до пяти значащих цифр действительных чисел. Тогда в учебной ячейке могут быть представлены числа из множества $\mathbb{R}_y = \{x \notin \mathbb{R}_5^* \mid 1{,}0000 \cdot 10^{-9} \le |x| \le 9{,}9999 \cdot 10^{+9}\}$.

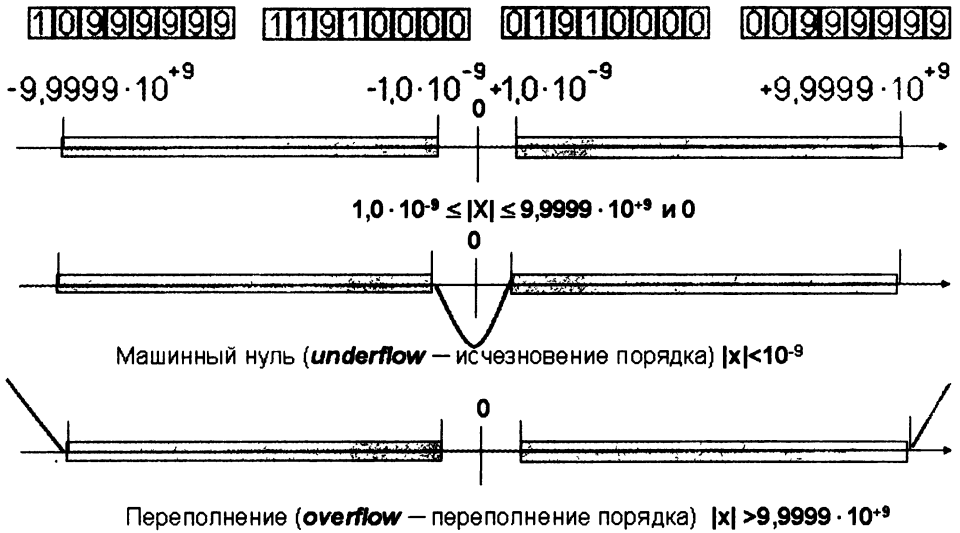


Рис. 2.11. Диапазоны представления чисел в формате с плавающей точкой в учебной ячейке

На рис. 2.11, *вверху*, приведены коды максимальных и минимальных по модулю чисел, их положение на числовой оси в *условном* масштабе, а также диапазоны чисел, представимых данным способом.

Из полученных соотношений следует, что число 0 закодировать данным способом невозможно, поскольку по выбранным условиям кодировки мантисса должна быть нормализована, $1 \le m < 10$. Это значит, что в четвертом разряде ячейки не должна находиться цифра нуль. Можно ослабить это требование и договориться о том, что заполненные нулями разряды с нулевого по пятый, то есть одновременно нулевой порядок и нулевая, ненормализованная мантисса, являются кодом числа нуль. Таким образом, число нуль в данной системе представлено четырьмя кодами: 00000000, 10000000, 01000000 и 11000000.

Обсудим теперь некоторые особенности выполнения действий над числами, принадлежащими множеству $\mathbb{R}_y$, коды которых, следовательно, могут быть записаны в учебной ячейке. Пусть $x_1 \notin \mathbb{R}_y$, $x_2 \in \mathbb{R}_y$, и результат выполнения какого-либо действия $\oplus$ над ними также принадлежит этому множеству: $x_1 \oplus x_2 \in \mathbb{R}_y$, где знак $\oplus$ обозначает любую арифметическую операцию. Тогда код результата также может быть записан в рассматриваемой ячейке. Особенности возникают в тех случаях, когда $x_1 \oplus x_2 \notin \mathbb{R}_y$, то есть когда результат выполнения каких-либо действий *не удовлетворяет* условию $1{,}0000 \cdot 10^{-9} \le |x| \le 9{,}9999 \cdot 10^{+9}$.

Пусть, например, умножаются нормализованные числа $5{,}0 \cdot 10^{-6}$ и $5{,}0 \cdot 10^{-8}$, каждое из которых может быть закодировано в обсуждаемой ячейке. Нормализованный результат равен $2{,}5 \cdot 10^{-13}$. Для сохранения модуля его порядка 13 требуется два разряда, в то время как в разрядной сетке для этого отведен только один разряд. Следовательно, обсуждаемым способом полученный результат не может быть закодирован. Ситуацию, когда в результате выполнения каких-либо действий получаются числа, у которых модуль меньше минимально возможного числа $|x| < 1 \cdot 10^{-9}$ (рис. 2.11, *посередине*), принято называть **машинным нулем**, **исчезновением порядка**, **потерей значимости** или **антипереполнением** (в англоязычной литературе используется термин underflow). При возникновении подобных ситуаций во время выполнения программ все числа из указанного диапазона считаются равными нулю, выдается соответствующее предупреждение или же выполнение программы прекращается.

Возможны нарушения ограничения и с другой стороны интервала. Пусть, например, умножаются нормализованные числа $5{,}0 \cdot 10^{+6}$ и $5{,}0 \cdot 10^{+8}$. Тогда нормализованный результат имеет вид $2{,}5 \cdot 10^{+15}$, и так же, как и в первом случае, он не может быть сохранен в обсуждаемой разрядной сетке. Ситуацию, когда в результате выполнения каких-либо действий получаются числа, модуль которых больше максимально возможного числа $|x| > 9{,}9999 \cdot 10^{+9}$ (рис. 2.11, *внизу*), принято называть **переполнением порядка**, а в англоязычной литературе — overflow. Эта ситуация более критическая, чем исчезновение порядка, так как простого выхода, аналогичного приравниванию результата к нулю, нет. Остается выдача соответствующего предупреждения или же прекращение выполнения программы. Кроме того, как и в случае исчезновения порядка, можно предусмотреть специальную реакцию программы на возникновение обсуждаемых исключительных ситуаций, например закрепление за результатом максимально или минимально возможного вещественного числа.

Обсуждая диапазон представимых в условной ячейке вещественных чисел, мы все время подчеркивали, что данные результаты получены для описанной ранее разрядной сетки, то есть для предложенной схемы закрепления разрядов ячейки за элементами кода. Изменение разрядной сетки существенно влияет на диапазон представимых чисел даже при неизменном общем количестве разрядов в ячейке. Для иллюстрации этого соображения вновь рассмотрим ту же самую учебную ячейку, состоящую из восьми десятичных разрядов. Но теперь под хранение кода порядка отведем два разряда, четвертый и пятый, а под мантиссу — оставшиеся четыре разряда, с нулевого по третий (см. рис. 2.9, *справа*). Другими словами,

при сохранении общего количества разрядов в ячейке увеличим количество разрядов, отводимых под код порядка, и одновременно уменьшим количество разрядов, отводимых под запись мантиссы. Как и в предыдущем случае, содержимое знаковых разрядов на рисунке показывает, что в ячейке закодировано положительное число с положительным порядком. Но теперь порядок равен содержимому четвертого и пятого разрядов, то есть 53, а мантисса находится в разрядах с нулевого по третий и равна 4,708. Получается, что теперь 00534708 — это код совершенно другого числа $+4{,}708 \cdot 10^{+53}$. Из этого результата можно сделать следующий важный вывод: интерпретация одного и того же кода зависит не только от используемого формата, но и от используемой разрядной сетки.

Сравнивая числа $+3{,}4708 \cdot 10^{+5}$ и $+4{,}708 \cdot 10^{+53}$ и учитывая при этом различия в соответствующих им разрядных сетках, можно, во-первых, увидеть, что, уменьшив количество разрядов, в которых записывалась мантисса, мы потеряли в точности задания числа. Если в первом случае число задавалось с точностью пять значащих цифр, то теперь значащих цифр только четыре. Во-вторых, увеличив количество разрядов, занятых кодом порядка, мы получили число $+4{,}708 \cdot 10^{+53}$, которое далеко вышло за границы диапазона $1{,}0000 \cdot 10^{-9} \le |x| \le 9{,}9999 \cdot 10^{+9}$. Рассуждая так же, как и в случае предыдущего варианта разрядной сетки, найдем, что в новой разрядной сетке представимы числа из диапазона $1{,}000 \cdot 10^{-99} \le |x| \le 9{,}999 \cdot 10^{+99}$, что существенно превышает первоначальные возможности.

ВНИМАНИЕ

При изменении количества разрядов поля, отводимых под код порядка, соответствующим образом изменяется диапазон представимых в ячейке чисел, а при изменении количества разрядов, отводимых под код мантиссы, соответствующим образом изменяется точность представления числа.

Стандарт IEEE 754

Ранее мы рассмотрели общие принципы кодирования вещественных данных в формате с плавающей точкой в *воображаемой* ячейке. Теперь перейдем к конкретным, используемым на практике представлениям этих данных в реальных полях памяти компьютера, состоящих из битов и из байтов, а не из *условных* десятичных разрядов.

В ранний период развития вычислительной техники разные разработчики компьютеров использовали различные аппаратные реализации формата с плавающей точкой. Его техническая реализация имеет много тонкостей, связанных с реакцией процессора на возникающие во время вычислений ситуации исчезновения порядка и переполнения. Неаккуратный подход к реализации иногда приводил к возникновению грубых ошибок в вычислениях. Кроме того, различные представления формата с плавающей точкой препятствовали переносу программ с компьютеров одной модели на компьютеры других моделей. Такой разнобой продолжался до начала 1980-х гг., когда одной из комиссий **IEEE** (Institute of Electrical and Electronics Engineers — Международный институт инженеров по электротехнике и электронике) было поручено создание стандарта для реализации формата

с плавающей точкой. В результате был создан стандарт на представление формата с плавающей точкой в компьютерах, который под номером **IEEE 754** был принят в 1985 г. в качестве международного. В настоящее время процессоры основных мировых разработчиков (Intel, SPARC, Power и т. д.) придерживаются этого стандарта.

Стандарт IEEE 754 предусматривает, что двоичные коды чисел в формате с плавающей точкой могут занимать поля длиной 4, 8 и 10 байтов. В соответствии с общими принципами кодирования чисел в этом формате поля содержат код знака числа, код порядка и его знака, а также код мантиссы.

Код знака числа как всегда занимает один, самый левый разряд поля. Кодируется знак также стандартным образом: 0 для положительных чисел и 1 для отрицательных.

Ранее было выяснено, что у нормализованных чисел порядки могут быть как положительными, так и отрицательными целыми числами. Следовательно, встает вопрос о кодировании в некоторой части поля знакового целого числа. Фактически, в примерах, приведенных на рис. 2.9–2.11, для кодировки порядка использована система кодирования со знаком, которая неприемлема для практической реализации по выясненной ранее причине — в силу возникновения для этой системы специальной арифметики. По ряду технических причин обратная и дополнительная системы также оказались неудобными для кодировки отдельной группы битов поля. Поэтому разработчики стандарта предложили использовать для кодирования порядка со знаком *систему со смещением*, константа которой равна $K = 2N^{-1} - 1$, где N — количество разрядов поля, выделенных под хранение порядка и его знака. Если истинный порядок нормализованного числа равен $P_и$, то в указанных битах поля записывается прямой двоичный (беззнаковый) код суммы $P_м = P_и + K$, который принято называть **машинным порядком** исходного вещественного числа. Количество разрядов N, которое выделяется под машинный порядок, равно 8, 11 и 15 для 4-, 8- и 10-байтовых полей соответственно.

ВНИМАНИЕ

В стандарте IEEE 754 минимальный $000...00_2$ и максимальный $111...11_2$ N-битовые машинные порядки играют особую роль. Они не служат для кодирования порядков обычных чисел, а используются в качестве признака особых, специальных значений, которые обеспечивают корректное выполнение операций даже при выходе результата за используемую разрядную сетку представления числа с плавающей точкой.

Таким образом, в обсуждаемом стандарте минимально возможный N-битовый машинный порядок у обычных чисел — $000...01_2$, а максимально возможный — $111...10_2$. Более подробно этот вопрос обсуждается в дальнейшем.

Оставшиеся от знакового бита и кода порядка разряды полей отводятся под код мантиссы, который, таким образом, занимает 23, 52 и 64 бита для 4-, 8- и 10-байтовых полей соответственно. В стандарте IEEE 754 мантисса *всегда* считается нормализованной, за исключением уже упоминавшихся специальных значений, которые имеют порядки $000...00_2$ и $111...11_2$. Но у нормализованной мантиссы

в двоичной системе счисления целая часть всегда равна единице. Поэтому разработчики стандарта предложили при хранении числа эту всегда равную единице целую часть не указывать в разрядной сетке поля. При хранении кодов чисел она только подразумевается, но во время выполнения действий целая часть автоматически восстанавливается. С помощь этого приема обеспечивается увеличение точности дробной части мантиссы, так как сэкономленный на неявно задаваемой целой части разряд отводится под дополнительную цифру дробной части мантиссы.

Пусть, например, нормализованная мантисса числа равна $1{,}011011111_2$ и под ее хранение выделено восемь битов. Тогда, если сохранять равную единице целую часть числа, при округлении необходимо отбросить *две* последние цифры дробной части и записать в поле код $1011\ 0111_2$, подразумевая при этом, что первая слева цифра является целой частью числа. Если в соответствии со стандартом целую часть не записывать в поле, то при округлении достаточно отбросить только *одну* цифру и записать в поле код $0110\ 1111_2$, подразумевая при этом, что первая слева цифра есть первая цифра дробной части числа, а равная единице целая часть в поле не указана.

Отметим, что в некоторых иностранных источниках мантиссой (mantissa) называются хранящиеся в поле значащие цифры дробной части $0110\ 1111_2$, а значение, содержащее и целую, и дробную часть, $1{,}01101111_2$, называют **сигнификантом** (от significant — значительный, существенный). В используемой же в учебнике терминологии $1{,}0110\ 1111_2$ — это мантисса, а $0110\ 1111_2$ — это ее код.

Таблица 2.5. Диапазоны и точности представления чисел в стандарте IEEE 754

Название	Длина (порядок, мантисса)	Min*	Min	Max	Точность
Real	32 (8, 23)	$1{,}4 \cdot 10^{-45}$	$1{,}17 \cdot 10^{-38}$	$3{,}4 \cdot 10^{+38}$	7–8
Double	64 (11, 52)	$5{,}0 \cdot 10^{-324}$	$2{,}2 \cdot 10^{-308}$	$1{,}8 \cdot 10^{+308}$	15–16
Extended	80 (15, 64)	$3{,}6 \cdot 10^{-4951}$	$3{,}4 \cdot 10^{-4932}$	$1{,}2 \cdot 10^{+4932}$	19–20

Рассмотрим более подробно кодирование чисел в формате с плавающей точкой для каждого из предусмотренных в стандарте IEEE 754 полей. В табл. 2.5 представлены используемые в языках программирования названия для типов данных, соответствующих стандартным полям, и их длины в битах. Во второй графе в скобках указано количество битов, отводимых под порядок со знаком и под мантиссу. Далее указаны диапазоны чисел, которые могут быть закодированы в этих полях, и заданные количеством значащих цифр точности чисел. Диапазоны ограничены приведенными в столбцах Min и Max наибольшим и наименьшим нормализованными числами, которые могут быть записаны в поле данной длины.

В языках программирования размещаемые в четырехбайтовых полях в формате с плавающей точкой числа относятся к типу real, их принято называть **короткими вещественными**, или **вещественными с одинарной точностью**. Разрядная сетка для этого поля приведена на рис. 2.12, *а*. Крайний левый, 31-й бит поля является знаковым. Порядок и его знак занимают 8 разрядов, с 23-го по 30-й. Константа

смещения K для кода порядка равна $2^7 - 1 = 127_{10} = 7F_{16} = 1111111_2$. Оставшиеся разряды с 0-го по 22-й содержат код мантиссы числа, при этом ее целая часть в поле не указывается.

Исходя из приведенной характеристики разрядной сетки можно определить точность и диапазон кодируемых в ней чисел. Для этого следует использовать известное соотношение $m \approx n \log_p q$, связывающее количество цифр в записях дробных частей числа в разных системах счисления с основаниями этих систем. В данном случае нужно определить количество разрядов m в десятичной системе счисления ($p = 10$), которыми будет изображаться мантисса, занимающая $n = 24$ разряда в двоичной системе ($q = 2$). Обратите внимание на то, что с учетом бита целой части занимающая 23 разряда мантисса фактически состоит из 24 бит. Далее элементарный расчет дает искомую оценку $m \approx 24 \log_{10} 2 \approx 24 \cdot 0{,}3010 \approx 7{,}22$. Таким образом, четырехбайтовое поле обеспечивает в десятичной системе точность 7–8 значащих цифр.

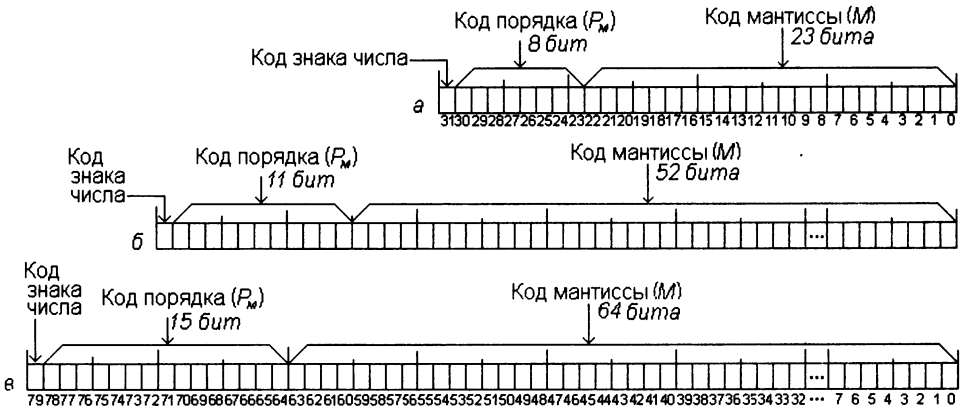


Рис. 2.12. Разрядные сетки стандарта IEEE 754 для формата с плавающей точкой

Займемся теперь оценкой диапазона кодируемых чисел. Максимальное число, которое может быть закодировано в данной сетке, найдем из следующих соображений. Максимально возможный машинный порядок представлен кодом 11111110_2, а максимально возможная мантисса состоит из 23 единиц $111..11_2$. Полагая знаковый бит равным нулю, получаем код наибольшего представимого в четырехбайтовом поле числа в виде $0|111\ 1111\ 0|111\ 1111\ 1111\ 1111\ 1111\ 1111_2$, или 7F 7F FF FF_{16}. Восстанавливая бит целой части, получим мантиссу $1{,}111\ 1111\ 1111\ 1111\ 1111\ 1111_2$. Если теперь прибавить к ней ненормализованное дробное число $0{,}000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = 2^{-23}{}_{10}$, то в сумме получим $10_2 = 2_{10}$. Таким образом, в десятичной системе счисления мантисса равна $2–2^{-23}$. Записывая максимальный машинный порядок в виде FE_{16}, получим истинный порядок $P_и = FE_{16} - 7F_{16} = 7F_{16} = 127_{10}$. Итак, наибольшее представимое в четырехбайтовом поле в формате с плавающей точкой число равно $(2 - 2^{-23}) \cdot 2^{+127} \approx \approx 2^{+128} \approx 3{,}4 \cdot 10^{+38}$.

Минимальное *нормализованное* число, которое можно закодировать в данной сетке, находим с помощью тех же самых рассуждений. Минимально возможный машинный порядок представлен кодом $0000\ 0001_2$, при этом наименьшая возможная мантисса состоит из 23 нулей, следовательно, код такого числа имеет вид $0|000\ 0000\ 1|\ 000\ 0000\ 0000\ 0000\ 0000\ 0000_2$, или $00\ 80\ 00\ 00_{16}$. Восстанавливая в мантиссе целую часть, получим ее исходное значение $1{,}0_2 = 1{,}0_{10}$, при этом истинный порядок равен $01_{16} - 7F_{16} = -7E_{16} = -126_{10}$. Итак, искомое нормализованное число равно $1{,}0 \cdot 2^{-126} = 1{,}17 \cdot 10^{-38}$.

Разрядная сетка стандартного поля длиной 8 байтов изображена на рис. 2.12, *б*. В языках программирования размещаемые в таких полях в формате с плавающей точкой числа относятся к типу double, их принято называть **длинными вещественными**, или вещественными **с двойной точностью**. Разряды этого поля распределяются следующим образом. Код знака числа занимает 63-й разряд, код машинного порядка $P_м$ занимает разряды с 52-го по 62-й, а под код мантиссы отведены разряды с 0-го по 51-й. Так как порядок занимает 11 битов, то константа смещения $K = 2^{10} - 1 = 1023_{10} = 3FF_{16} = 111\ 1111\ 1111_2$. Точность чисел в десятичной системе равна $m \approx 52 \log_{10} 2 \approx 52 \cdot 0{,}3010 \approx 15{,}65$, или 15–16 значащих цифр. Код максимального числа $7F\ EF\ FF\ FF\ FF\ FF\ FF\ FF_{16}$, при этом машинный порядок числа равен $7FE_{16}$, а истинный — $7FE_{16} - 3FF_{16} = 3FF_{16} = 1023_{10}$. Тем же самым способом можно получить мантиссу наибольшего числа: $2 - 2^{-52}$. Следовательно, наибольшее представимое в восьмибайтовом поле в формате с плавающей точкой число равно $(2 - 2^{-52}) \cdot 2^{+1023} \approx 2^{+1024} \approx 1{,}798 \cdot 10^{+308}$. Код минимального нормализованного числа в восьмибайтовом поле имеет вид $00\ 10\ 00\ 00\ 00\ 00\ 00\ 00_{16}$, его мантисса равна $1{,}0_2 = 1{,}0_{10}$, а истинный порядок — $001_{16} - 3FF_{16} = -3FE_{16} = -1022_{10}$. Отсюда наименьшее нормализованное число равно $1{,}0 \cdot 2^{-1022} = 2{,}2 \cdot 10^{-308}$.

Последнее стандартное поле для представления чисел в формате с плавающей точкой имеет длину 10 байтов (рис. 2.12, *в*). В языках программирования размещаемые в таких полях в формате с плавающей точкой числа относятся к типу extended, их принято называть **расширенными вещественными**. Разряды поля распределены следующим образом. Код знака числа занимает 79-й разряд, код машинного порядка $P_м$ занимает разряды с 64-го по 78-й, а под код мантиссы отведены разряды с 0-го по 63-й. Константа смещения K для данного поля равна $2^{14} - 1 = 16383_{10} = 3F\ FF_{16} = 11\ 1111\ 1111_2$.

При кодировании мантиссы в десятибайтовом поле, в отличие от кодирования в двух других полях, ее целая часть *не отбрасывается*, а указывается в отведенных под код мантиссы битах поля вместе со всеми остальными цифрами дробной части. Это вызвано тем обстоятельством, что десятибайтовая разрядная сетка используется не только для хранения вещественных чисел, но и для выполнения над ними различных действий. Если первоначально число в формате с плавающей точкой хранится в четырех- или восьмибайтовом поле, то перед выполнением какого-либо действия в коде мантиссы этого числа восстанавливается единица целой части и производится расширение кода до десятибайтовой сетки. Таким приемом обеспечиваются правильность выполнения всех арифметических операций над числами в формате с плавающей точкой и наивысшая возможная точность вычислений.

С учетом указанного отличия получим параметры представления вещественных чисел для десятибайтового поля. Точность в десятичной системе равна $m \approx 64 \log_{10} 2 \approx 64 \cdot 0,3010 \approx 19,26$, или 19–20 значащих цифр. Код максимального числа $7F\ FE\ FF\ FF\ FF\ FF\ FF\ FF\ FF\ FF_{16}$, при этом его машинный порядок равен $7F\ FE_{16}$, а истинный — $7FFE_{16} - 3FFF_{16} = 3FFF_{16} = 16383_{10}$. Мантисса наибольшего числа равна $2 - 2^{-63}$. Следовательно, наибольшее представимое в 10-байтовом поле в формате с плавающей точкой число равно $(2 - 2^{-63}) \cdot 2^{+16383} \approx 2^{+16384} \approx$ $\approx 1,19 \cdot 10^{+4932}$. Код минимального нормализованного числа в десятибайтовом поле имеет вид $00\ 01\ 80\ 00\ 00\ 00\ 00\ 00\ 00\ 00_{16}$, его мантисса равна $1,0_2 = 1,0_{10}$, а истинный порядок — $0001_{16} - 3FFF_{16} = -3FFE_{16} = -16382_{10}$. Отсюда наименьшее нормализованное число равно $1,0 \cdot 2^{-16382} = 3,36 \cdot 10^{-4932}$.

ВНИМАНИЕ

Сравнивая точность представления чисел в полях различной длины, легко заметить, что точность чисел, определяемая как количество значащих цифр числа в десятичной системе счисления, равна удвоенной длине поля в байтах.

Получение кода числа в формате с плавающей точкой. Чтобы получить машинный код числа в формате с плавающей точкой, следует:

1) перевести модуль числа из десятичной системы счисления в двоичную;
2) нормализовать число;
3) сформировать знаковый бит;
4) выделить истинный порядок $P_и$;
5) получить код машинного порядка $P_м = P_и + K$;
6) для четырех- и восьмибайтовых полей получить код мантиссы M, отбрасывая бит целой части, для десятибайтового поля этот бит не отбрасывается;
7) записать код знака, код порядка $P_м$ и код мантиссы M в отведенные для них позиции разрядной сетки.

Найдем, например, код числа $-15,375_{10}$ в формате с плавающей точкой в четырехбайтовом поле. Перевод этого числа в двоичную систему счисления дает в результате $-1111,011_2$. Для нормализации числа запятую нужно сместить на три позиции влево. Чтобы значение числа при этом не изменилось, его следует умножить на основание системы счисления в третьей степени. Таким образом, нормализованное число имеет вид $1,111011 \cdot 10_2^{+11}$. Обратите внимание на то, что порядок $+3_{10}$ записан как $+11_2$. Далее получаем код знака числа 1 и его истинный порядок $P_и = 3_{10} = 3_{16} = 11_2$. Код машинного порядка имеет вид $P_м = 11_2 + 111\ 1111_2 = 1000\ 0010_2$. Если у мантиссы нормализованного представления числа $1,111011_2$ отбросить целую часть (вместе с запятой), то получится код мантиссы $M = 111011_2$. Собирая вместе все элементы кода и заполняя незначащие правые биты мантиссы нулями, получим искомый код числа в виде $1|10000010|11101100000000000000000_2$. Для наглядности в записи кода вертикальной чертой отделены друг от друга биты знака, порядка и мантиссы. Следу-

ет понимать, что в реальных машинных кодах таких разделительных знаков нет. Итак, код числа $-15{,}375_{10}$ в формате с плавающей точкой в четырехбайтовом поле имеет вид 1100 0001 0111 0110 0000 0000 0000 0000_2, или в компактном шестнадцатеричном виде C1 76 00 00_{16}.

Не останавливаясь на деталях выполнения действий, приведем шестнадцатеричные коды числа $-15{,}375_{10}$ в восьмибайтовом (C0 2E C0 00 00 00 00 00_{16}) и в десятибайтовом (C0 02 F6 00 00 00 00 00 00 00_{16}) полях. Сравнивая эти коды друг с другом и с предыдущим результатом, отметим, что переход к более длинному полю для формата с плавающей точкой *не сводится* к заполнению дополнительных битов нулем или единицей, как это было в случае формата с фиксированной точкой. Более существенные различия в кодах вызваны различным количеством позиций, отводимых под код порядка, и, соответственно, другой константой смещения, а также различиями в способе кодирования мантиссы.

Получение числа по его коду в формате с плавающей точкой. Чтобы определить значение числа, для которого задан его машинный код в формате с плавающей точкой, нужно:

1) получить запись кода в двоичной системе счисления;
2) выделить код знака и определить знак числа;
3) выделить код машинного порядка $P_м$;
4) по формуле $P_и = P_м - K$ получить истинный порядок $P_и$;
5) выделить код мантиссы M и для четырех- и восьмибайтовых полей получить мантиссу, дописывая бит целой части; для десятибайтового поля мантисса совпадает со своим кодом;
6) получить запись числа в нормализованной форме в двоичной системе счисления;
7) при необходимости денормализовать число, затем перевести его в десятичную систему счисления и сформировать нужный знак.

Пусть, например, задан код C2 B7 AE 00_{16}. Его двоичная запись имеет вид 1100 0010 1011 0111 1010 1110 0000 0000_2. Последующее выделение элементов кода можно осуществить, отделив после крайнего слева содержащего единицу знакового бита еще восемь битов $1000\ 0101_2$ кода порядка. Оставшиеся биты 0110 1111 0101 1100 000 0000_2 представляют собой код мантиссы.

Равный единице знаковый бит свидетельствует о том, что в поле находится код отрицательного числа. На основании кода машинного порядка $P_м = 1000\ 0101_2 = 85_{16}$ получаем истинный порядок числа: $P_и = 85_{16} - 7F_{16} = 6_{16} = 110_2$. Восстанавливая мантиссу из ее кода, получаем $1{,}01101111010111_2$. Собирая вместе полученные элементы числа, находим его нормализованную запись в виде $1{,}01101111010111 \cdot 10^{+110}$. Смещая далее запятую направо на шесть позиций, получим ненормализованную запись числа в двоичной системе счисления: $1011011{,}11010111_2$. Отсюда уже просто получить его значение в шестнадцатеричном ($-5B{,}D7_{16}$), а затем и в десятичном ($-91{,}83984375_{10}$) виде.

Специальные значения стандарта IEEE 754

Как уже отмечалось, в стандарте IEEE 754 имеются средства, которые позволяют получать правильные результаты или результаты с небольшой ошибкой даже в случае выхода полученного во время вычислений значения за границы разрядной сетки. Для этого предусмотрено формирование в полях *специальных значений*, которые образуются с помощью нулевого $000...00_2$ и максимального $111...11_2$ N-битовых порядков. К специальным значениям относятся:

- денормализованные вещественные числа;
- плюс и минус нуль;
- плюс и минус бесконечность;
- нечисла;
- неопределенности.

В табл. 2.6 приведены сводка специальных значений стандарта и отличительные признаки их кодов.

Таблица 2.6. Специальные значения стандарта IEEE 754

Специальное значение	Признак
Денормализованное число	Любой знак, нулевой код машинного порядка $000...00_2$ и любой ненулевой код мантиссы
Нуль	Любой знак, нулевой код машинного порядка $000...00_2$ и нулевой код мантиссы
Бесконечность	Любой знак, код порядка из одних единиц $111...11_2$ и нулевой код мантиссы
Нечисло, NAN	Любой знак, код порядка из одних единиц $111...11_2$ и любой ненулевой код мантиссы
Неопределенность	Код знака «минус», код порядка из одних единиц $111...11_2$, первая цифра дробной части мантиссы единица, остальные нули $1{,}1000...00_2$

Большую часть специальных значений можно использовать в вычислениях наряду с обычными числами. Это дает возможность разрабатывать программы, устойчивые к появлению ошибочных ситуаций.

Для повышения точности вычислений при работе с маленькими числами в стандарте предусмотрена возможность использования так называемого **мягкого исчезновения порядка (мягкого антипереполнения)**. Суть этого подхода состоит в трактовке кодов с нулевым порядком $000...00_2$ как специальных значений, которые принято называть **денормализованными числами**. При этом используется следующее правило: если код содержит все нули в отведенных под порядок разрядах $000...00_2$ и ненулевую мантиссу, то считается, что порядок числа равен $000...001_2$, а явно не указанная в поле целая часть мантиссы принимается равной нулю.

Таблица 2.7. Мягкое исчезновение порядка

Код числа	000000001100...0	000000001000...0	000000000100...0	000000000010...0
Код порядка	00000001	000000001	00000000	00000000
Порядок	00000001	000000001	00000001	00000001
Код мантиссы	1000000...0000000	0000000...0000000	1000000...000000	0100000...000000
Неявный бит	1	1	0	0
Мантисса	1,1	1,0	0,1	0,01
Значение	$1{,}1 \cdot 2^{-126} \approx \approx 1{,}3 \cdot 10^{-38}$	$1{,}0 \cdot 2^{-126} \approx \approx 1{,}2 \cdot 10^{-38}$	$0{,}1 \cdot 2^{-126} \approx \approx 5{,}9 \cdot 10^{-39}$	$0{,}1 \cdot 2^{-127} \approx \approx 7{,}32 \cdot 10^{-40}$

Проиллюстрируем применение этого правила на нескольких примерах. В верхней строке табл. 2.7 приведены коды чисел в четырехбайтовом поле в формате с плавающей точкой. Далее находятся код порядка, полученный на его основании порядок, код мантиссы, неявный бит целой части мантиссы, мантисса и соответствующее коду значение числа.

Второй и третий столбцы таблицы содержат отличные от нуля коды порядка. Следовательно, в полях находятся обычные числа, неявный бит целой части равен единице и значения чисел определяются по общим правилам. Выполняя расчеты, получим, что третий столбец занимает код минимального числа $1{,}0 \cdot 2^{-126} \approx 1{,}17 \cdot 10^{-38}$, а второй — код предшествующего ему большего числа $1{,}1 \cdot 2^{-126} \approx 1{,}3 \cdot 10^{-38}$. В четвертом и пятом столбцах код порядка равен нулю, 00000000_2. Из этого следует, что в полях находятся специальные значения, для которых действуют особые правила определения значений чисел. Для них порядок считается равным минимально возможному $0000\ 0001_2$, а неявный бит целой части считается равным нулю. Следовательно, мантисса *ненормализованная*, и ее код фактически совпадает с числом. Применяя эти правила к коду в четвертом столбце, получим число $0{,}1 \cdot 2^{-126} \approx 5{,}9 \cdot 10^{-39}$, а код из пятого столбца даст число $0{,}01 \cdot 2^{-126} \approx 0{,}1 \cdot 2^{-127} \approx 7{,}32 \cdot 10^{-40}$.

Очевидно, что последовательное, разряд за разрядом, перемещение направо положения первой значащей единицы в мантиссе приводит к постепенному уменьшению кодируемого значения. Последним в этом процессе является код $0|00000000|00000000000000000000001_2$ с нулевым порядком и ненулевой мантиссой $0{,}1 \cdot 2^{-22}$, который соответствует числу со значением $1{,}0 \cdot 2^{-23} \cdot 2^{-126} \approx 1{,}40 \cdot 10^{-45}$. Таким образом, вместо стандартного резкого перехода от наименьшего *нормализованного* числа к нулевому значению, которое считается округлением всех чисел с порядком, меньшим минимально возможного, в данном подходе диапазон порядков постепенно, «мягко» расширяется в отрицательную сторону за счет уменьшения точности мантиссы.

Рассмотрим пример, когда использование мягкого исчезновения порядка позволяет получить правильный результат, в то время как резкий переход к нулю дает

ошибку. Пусть даны два числа x и y, требуется получить разность $y - x$, как-то ее использовать, а затем вернуться к исходному значению y, то есть затем фактически нужно вычислить $(y - x) + x$. Если разность $y - x$ принадлежит множеству допустимых для используемого поля значений, то проблем не возникает. В противном случае могут появиться ошибки. Пусть, например, получены значения $y = 1{,}5 \cdot 10^{-38}$, $x = 1{,}2 \cdot 10^{-38}$. В соответствии с табл. 2.5 оба эти числа могут быть закодированы в четырехбайтовом поле, но их разность $y - x$, равная $0{,}3 \cdot 10^{-38}$, уже выходит за границы разрядной сетки. Если в компьютере реализовано стандартное резкое исчезновение порядка, то эта разность признается машинным нулем, $y - x = 0$, и конечный результат окажется ошибочным, так как сложение нуля с числом x даст в результате x. Если же в компьютере реализован подход стандарта IEEE 754 с мягким исчезновением порядка, то вычисление разности $y - x$ приведет к *специальному денормализованному* значению $0{,}3 \cdot 10^{38}$, дальнейшее сложение которого с числом x даст правильный результат $y = 1{,}5 \cdot 10^{-38}$.

Выполняя соответствующие расчеты для восьмибайтовых и десятибайтовых полей, найдем, что для них предельные денормализованные значения равны $1{,}0 \cdot 2^{-52} \cdot 2^{-1022} \approx 5{,}18 \cdot 10^{-318}$ и $1{,}0 \cdot 2^{-63} \cdot 2^{-16382} \approx 3{,}64 \cdot 10^{-4951}$ соответственно. Именно эти значения приведены в столбце Min* в табл. 2.5 как минимально возможные *ненормализованные* числа для полей соответствующей длины.

Дальнейшее уменьшение кода мантиссы приводит к коду с нулевым порядком и нулевой мантиссой, который также относится к специальным значениям и по определению считается кодом нуля. В формате с плавающей точкой допускается наличие двух закрепленных за нулем кодов: кода $00\ 00\ 00\ 00_{16}$, соответствующего числу $+0_{10}$, и кода $80\ 00\ 00\ 00_{16}$, соответствующего числу -0_{10}. Такие же специальные коды предусмотрены и для восьми- и десятибайтовых полей. В отличие от кодирования целого нуля, то есть нуля в формате с фиксированной точкой, наличие двух кодов у вещественного нуля, то есть нуля в формате с плавающей точкой, в силу приближенного характера вычислений и концепции машинного нуля не вызывает сложностей в расчетах.

Ранее обсуждались предусмотренные в стандарте меры, обеспечивающие выполнение расчетов в случае исчезновения порядка, то есть когда порядок становится меньше, чем минимально возможный для данной разрядной сетки. Эти возможности базируются на специальной интерпретации кодов с нулевым машинным порядком $000...00_2$. К сожалению, организовать нечто подобное, то есть какой-либо плавный выход за разрядную сетку в сторону больших порядков невозможно. Поэтому разработчики стандарта предложили рассматривать коды с машинными порядками, равными $111...11_2$, и с равной нулю мантиссой как специальные значения, имеющие смысл *плюс или минус бесконечности* в зависимости от кода знака. Итак, находящийся в четырехбайтовом поле код $0111\ 1111\ 1000\ 0...00_2$ или $7F\ 80\ 00\ 00_{16}$, считается кодом плюс бесконечности ($+\infty$), а код $1111\ 1111\ 1000\ 0...00_2$ или $FF\ 80\ 00\ 00_{16}$ считается кодом минус бесконечности ($-\infty$). Такие же специальные значения предусмотрены для восьми- и десятибайтовых полей. Знаковые нули и знаковые бесконечности могут использоваться в расчетах как вместе с обычными, то есть нормализованными, так

и с денормализованными числами. При этом бесконечности трактуются в *аффинном* смысле, то есть для любого конечного числа x верно неравенство $-\infty < x < +\infty$. Действия, использующие коды конечных чисел x, коды нулей +0 и –0, а также коды бесконечностей, приводят к естественным результатам. Для определенности будем считать, что $x > 0$, тогда $x/(+\infty) = +0$, $x/(-\infty) = -0$, $+\infty \cdot x = +\infty$, $-\infty \cdot x = -\infty$, $(+\infty) \cdot (+\infty) = +\infty$ и т. д.

Однако операции типа ∞/∞ и $0/0$, как и в обычном анализе, не имеют смысла. Чтобы обеспечить возможность продолжения вычислений с получением результатов, которые в таких случаях все-таки могут каким-то образом трактоваться, в стандарте IEEE 754 введены специальные значения **нечисло — NaN** (от Not a Number). Код нечисла может иметь любой знак, машинный порядок должен состоять только из единиц $111...11_2$, а код мантиссы может быть любым ненулевым кодом. Следовательно, существует много кодов различных нечисел. За различными нечислами разработчики компьютеров и программисты могут закреплять различный смысл и при их появлении в вычислениях делать соответствующие выводы. В частности, нечисло, имеющее знак «минус» и мантиссу 1,1000...000 (с первой значащей цифрой дробной части равной единице и всеми остальными нулями), считается *неопределенностью*, возникающей при делении на нуль.

Сравнивая логику вычислений с данными, представленными в формате с фиксированной точкой, и логику выполнения действий над данными в формате с плавающей точкой, несложно заметить существенно более сложный характер вычислений с плавающей точкой. В последнем случае необходимо, по крайней мере, уметь выделять одиночные биты и некоторые группы битов — участки полей, которые соответствуют кодам порядка и мантиссы. Дальнейшая раздельная обработка этих кодов уже близка к действиям над целочисленными данными. Для выделения некоторых групп битов процессор должен в дополнение к вышеперечисленным операциям сравнения, сложения и сдвига «уметь» выполнять над двоичными кодами такие операции, как дизъюнкция, конъюнкция и отрицание.

ВНИМАНИЕ

Минимальный набор действий, которым должен обладать процессор компьютера, включает операции сравнения, сложения, дизъюнкции и конъюнкции, а также пересылки, отрицания и сдвига одиночного кода на некоторое количество разрядов вправо и влево.

Как мы увидим в дальнейшем, именно этот набор действий совместно с операциями, обеспечивающими организацию ветвлений, циклов, вызов подпрограмм и т. д., образует основу системы команд любого процессора.

2.3.4. Особенности компьютерной арифметики

Завершая обсуждение принципов кодирования числовой информации, необходимо особо подчеркнуть, что, несмотря на используемые в современных компьютерах изощренные приемы кодирования, достичь полного соответствия между

обычной арифметикой и компьютерной обработкой числовых данных *принципиально невозможно*. Этот вывод является следствием того, что в любом случае код числа в памяти компьютера, представляющего собой реальное техническое устройство, занимает конечное количество разрядов поля. Это количество теоретически можно сделать сколь угодно большим, но оно принципиально не может быть бесконечным, в то время как в математике числа в общем случае изображаются бесконечным количеством значащих цифр.

Конечность количества разрядов поля, используемого для записи кода числа, приводит, во-первых, к *ограниченности диапазонов* кодируемых целых и вещественных чисел, а во-вторых, к конечной точности представления вещественных чисел. Если результат выполнения некоторой операции над данными с фиксированной точкой выйдет за границы диапазона представимых в данном поле чисел, то либо лишние биты окажутся утерянными, либо будет сформировано сообщение о возникновении ошибки и вычисления будут остановлены. Если для представления чисел используется формат с плавающей точкой, то результат может быть заменен машинным нулем или бесконечностью. В этом случае в зависимости от конкретной ситуации дальнейшие вычисления могут вернуть приемлемый приближенный результат либо так же, как и в предыдущем случае, может быть сформировано сообщение об ошибке с прекращением вычислений.

Указанные факторы влекут за собой также невыполнение в компьютерной арифметике ассоциативного и дистрибутивного законов обычной арифметики. Нарушение этих законов, в свою очередь, является причиной ненулевой вероятности возникновения ошибок в тех или иных конкретных расчетах.

Пример нарушения ассоциативного закона в компьютерной арифметике мы рассматривали в предыдущем разделе учебника. С точки зрения обычной арифметики, значение выражения $(y - x) + x$ всегда одно и то же и равно y, в то время как получаемый на компьютере результат существенно зависит от значений величин x и y, от используемого формата кодирования — с фиксированной или с плавающей точкой, а если используется формат с плавающей точкой, то и от реализованной в компьютере реакции на исчезновение порядка.

Теперь рассмотрим пример нарушения дистрибутивного закона. Пусть требуется вычислить значение выражения $(x + y)z$. Если значения x и y велики, то их суммирование может вызвать переполнение, и тогда результат не будет получен вообще или же он может трактоваться как бесконечно большой. С другой стороны, если сомножитель z мал и выражение реализовано в эквивалентном с точки зрения обычной арифметике виде $xz + yz$, то велика вероятность того, что вычисления вернут правильный результат. Итак, мы опять сталкиваемся с ситуацией, когда эквивалентные с точки зрения обычной математики выражения в компьютерном варианте вычислений *могут иметь разные значения*.

Известно, что на числовой прямой любой отрезок или интервал содержит несчетное множество действительных чисел, в то время как любое поле для представления чисел в формате с плавающей точкой всегда может быть использова-

но для записи лишь конечного количество различных кодов. Это влечет за собой следующую особенность, связанную с отсутствием свойства плотности у множества чисел с плавающей точкой. Свойство плотности множества действительных чисел состоит в том, что для любых двух действительных чисел α и β существует число χ — такое, что $\alpha < \chi < \beta$. В компьютерной математике из-за отсутствия этого свойства между двумя кодами не всегда можно найти код, обладающий таким свойством. И если в результате вычислений получается не представимое в разрядной сетке число, то его приходится округлять в ту или иную сторону. Соответственно на результат начинает влиять выбранный в компьютере способ округления, и при определенных обстоятельствах ошибки округления могут накапливаться, существенно корректируя окончательный результат.

Указанные различия между обычной и компьютерной арифметикой необходимо всегда учитывать при разработке и отладке программ. В тех случаях, когда от программы требуется повышенная надежность, в нее слéдует включать все необходимые проверки, позволяющие гарантированно застраховаться от такого рода ошибок.

2.4. Звуковые и видеоданные

Развитие способов кодирования звуковой информации, а также движущихся изображений — **анимации** и видеозаписей — происходило с запаздыванием относительно развития способов кодирования рассмотренных выше разновидностей информации. Приемлемые способы хранения и воспроизведения с помощью компьютера звуковых и видеозаписей появились только в 1990-х годах. Эти способы работы со звуком и видео получили название **мультимедийных технологий**.

Звук представляет собой довольно сложное, непрерывное колебание воздуха. Основной подход к кодированию звука, который называется **преобразованием в цифровую форму**, **оцифровыванием** или **дискретизацией**, основан на том, что непрерывный звуковой сигнал заменяется дискретным (то есть состоящим из обособленных, раздельных элементов) набором значений сигнала — **отсчетов сигнала** — в некоторые последовательные моменты времени (рис. 2.13). Каждый отсчет кодируется в одном из рассмотренных ранее числовых форматов. Таким образом, кодирование и обработка звуковых данных фактически сводятся к кодированию и обработке числовых данных.

ВНИМАНИЕ

Пусть какое-либо событие происходит периодически через равные промежутки времени T. Тогда количество событий в единицу времени $1/T$ называется частотой события. Частота измеряется в герцах (Гц), при этом 1 Гц — это одно событие в секунду. Часто используемыми кратными единицами частоты являются килогерцы (1 кГц = 10^3 Гц), мегагерцы (1 МГц = 10^6 Гц) и гигагерцы (1 ГГц = 10^9 Гц).

В соответствии с этим определением количество отсчетов сигнала в единицу времени называется **частотой дискретизации**. Так, на рис. 2.13 сигнал, длящийся 2 с, заменяется 100 отсчетами (жирные точки на графике амплитуды сигнала). Следовательно, в данном случае частота дискретизации равна 50 Гц. При записи звука в мультимедийных технологиях применяются частоты 8, 11, 22 и 44 кГц. Частота дискретизации 44 кГц означает, что одна секунда непрерывного звучания заменяется набором из сорока четырех тысяч отдельных отсчетов сигнала. Чем выше частота дискретизации, тем лучше качество оцифрованного звука. В последних разработках частота дискретизации достигает 192 кГц.

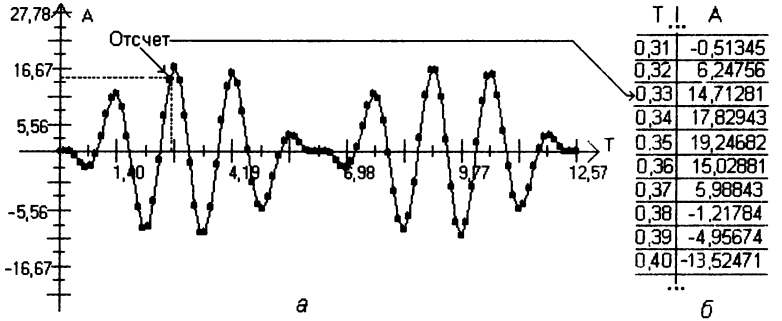


Рис. 2.13. Переход от аналоговой (непрерывной) формы сигнала (*а*) к цифровой (дискретной) (*б*)

Каждый отдельный отсчет представляет собой число, которое затем можно представить в виде некоторого двоичного кода. Качество преобразования звука в цифровую форму определяется не только частотой дискретизации, но и количеством битов памяти, отводимых на запись кода одного отсчета. Этот параметр принято называть **разрядностью преобразования**. В настоящее время обычно используются разрядности 8, 16 и 24 бит. На описанных принципах основывается формат **WAV** (от WAVeform-audio — волновая форма аудио) кодирования звука. Получить запись звука в этом формате можно от подключаемых к компьютеру микрофона, проигрывателя, магнитофона, телевизора и других стандартно используемых устройств работы со звуком. Однако формат WAV требует очень много памяти. Так, при записи стереофонического звука с параметрами, дающими хорошее качество звучания, частотой дискретизации 44 кГц и разрядностью 16 бит на одну минуту записи требуется примерно 10 Мбайт памяти.

Кодирование видеоинформации — еще более сложная проблема, чем кодирование звуковой информации, так как нужно позаботиться не только о дискретизации непрерывных движений, но и о синхронизации изображения со звуковым сопровождением. В настоящее время для этого используется формат, которой называется **AVI** (Audio-Video Interleaved — чередующиеся аудио и видео). Отметим, что основные мультимедийные форматы AVI и WAV очень требовательны к памяти. Поэтому на практике применяются различные способы компрессии (то есть сжатия) звуковых и видеокодов.

2.5. Принцип обезличивания кода

Перед завершением обсуждения общих принципов кодирования информации хотелось бы обратить внимание на один важный момент.

ВНИМАНИЕ

Для компьютера не существует разницы между кодами данных различной природы.

Возьмем какой-либо двоичный код, например $1000\ 1100_2$. Если обратиться к приведенному ранее фрагменту кодовой таблицы (см. табл. 2.2), то можно утверждать, что это код буквы М. В то же время можно сказать, что этим кодом задается цвет одного из пикселов монохромного изображения. Если предположить, что для кодирования использован беззнаковый формат с фиксированной точкой, то можно сказать, что это код числа 140_{10}. Предположение об использовании знакового представления в этом же формате приведет нас к выводу о том, что в поле находится код числа -120_{10}. Что же представляет собой этот код на самом деле? *Интерпретация (то есть истолкование смысла) машинного кода может быть самой разной.* Один и тот же код разными программами может рассматриваться и как число, и как текст, и как изображение, и как звук. Другими словами, как именно трактуется, понимается тот или иной машинный код, определяется обрабатывающей этот код программой.

На рис. 2.14 приведен ряд примеров возможных интерпретаций кода $C2ABE452_{16}$.

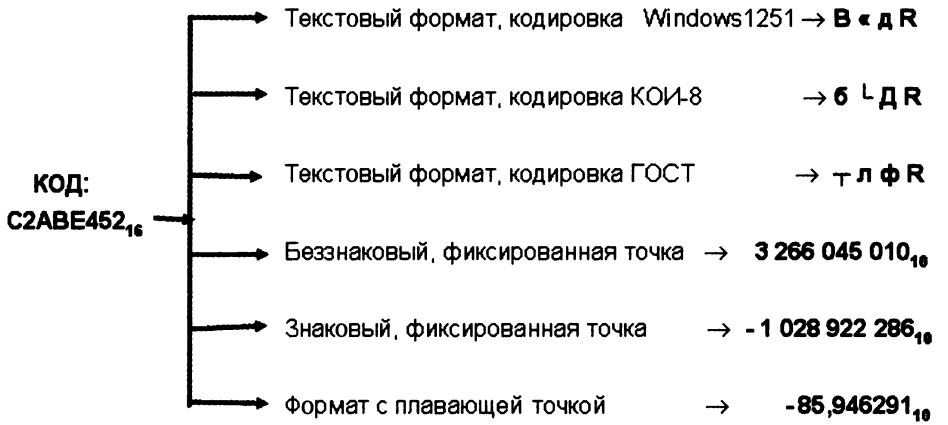


Рис. 2.14. Возможные интерпретации одного и того же двоичного кода

ВНИМАНИЕ

Принцип обезличивания машинного кода состоит в том, что один и тот же код может восприниматься процессором компьютера в любом из используемых в аппаратных и программных средствах способов кодирования. Конкретная трактовка кода определяется обращающейся к коду программой.

2.6. Надежность кодирования данных

Одним из важнейших аспектов организации хранения и передачи кодов данных является обеспечение их надежности и безошибочности. Существует множество различных факторов, воздействие которых на аппаратные средства компьютера приводит к искажению кодов данных. К ним относятся, например, влияние сильных электромагнитных полей от источников, находящихся вблизи запоминающих, передающих устройств и линий связи, влияние температурного расширения материалов, из которых состоят запоминающие и передающие устройства, сбои в работе аппаратуры, механические удары и т. д. Для защиты данных от большинства таких факторов к настоящему времени уже разработаны и внедрены достаточно надежные средства.

К сожалению имеются и факторы, от которых сложно и даже невозможно защититься. К таким факторам, например, относится так называемое α-излучение. Дело в том, что корпуса элементов памяти компьютера содержат атомы тяжелых элементов, которые вследствие радиоактивного распада испускают α-частицы. Попадая в собственно элементы памяти, они вызывают изменение их состояния с вероятностью в один случай за миллион лет. Память объемом 1 Мбайт содержит 8 388 608 бит, таким образом, среднее время появления ошибки для этого объема составляет примерно 1/8 года, или 45 дней. Если учесть, что в стандартной комплектации современных компьютеров объем оперативной памяти не менее 128 Мбайт, то этот фактор приводит к искажению в среднем *трех битов в день*. Такая высокая вероятность появления ошибки в кодах данных уже не является допустимой. На сегодняшний день не существует экономически оправданных технических способов защиты памяти от α-излучения. Неустранимые проблемы искажения информации возникают и при хранении данных на магнитных и оптических дисках с высокой плотностью, а также при их передаче по линиям связи с высокими посторонними шумами.

Вместе с тем, применяя специальные методы кодирования, можно обеспечить контроль за появлением ошибки и даже восстановление исходного кода после ее обнаружения. Применение этих методов позволяет понизить вероятность появления ошибки до 10^{-9} и ниже.

Общим методом обеспечения надежности хранения и передачи данных в компьютере и по линиям связи является включение в код *дополнительных контрольных разрядов* (битов). Существует много различных вариантов этого метода. Подробное их рассмотрение в рамках данного учебника невозможно, поэтому далее обсуждаются только наиболее распространенные способы.

Во многих компьютерах одним из уровней контроля за появлением ошибок является включение в байт дополнительного, *контрольного* девятого разряда, который формируется аппаратурой автоматически. А те восемь битов, из которых, как мы до сих пор считали, состоит байт, в дальнейшем будем называть **информационными**. Значение контрольного разряда определяется так, чтобы общее количество разрядов байта (всех информационных и контрольного), которые

содержат 1, было нечетным. Рассмотрим, например, байт с информационными битами 1001 1110_2. Количество единиц в нем нечетно, следовательно, аппаратура сформирует значение контрольного бита, равное 0. При этом общее количество единиц в девяти битах байта останется нечетным. А, например, для байта с информационными битами 1001 0110_2, количество единиц в котором четное, контрольный бит окажется равным 1 — в результате образуется нужное нечетное количество единиц.

На рис. 2.15 приведены примеры байтов с отмеченным буквой К контрольным битом. Байты на рис. 2.15, *а* не содержат ошибок. Контрольный бит содержит 0 или 1, так что общее количество единиц нечетное. В то же время байты на рис. 2.15, *б* содержат ошибочные коды.

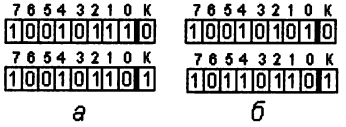


Рис. 2.15. Байты с контрольными битами: *а* — без ошибок; *б* — содержащие ошибку

Процессор во время выполнения любых действий контролирует общее количество единиц в байте по всем девяти битам и в случае нарушения требования его нечетности либо блокирует дальнейшую работу, либо принимает специальные меры по восстановлению верного значения кода, выполняя, например, повторную передачу или организуя повторное выполнение действия, приведшего к появлению ошибки. Отметим, что обычно используемая проверка на нечетность количества единиц в коде не является единственно возможной, с равным успехом можно использовать и проверку на четность этого количества. Еще раз подчеркнем, что введение одного контрольного разряда позволяет установить только факт появления ошибочного бита где-то в байте, но его положение (номер разряда) определить невозможно.

Теоретические соображения показывают, что ошибки могут быть не только одиночными. Существует ненулевая вероятность появления кратных ошибок, когда одновременно появляется два и больше сбойных битов в одном байте. В связи с этим следует уточнить возможности одиночного контрольного разряда. Если суммарное количество единиц в байте при использовании одного контрольного разряда оказалось четным, то можно утверждать только, что имеется нечетное (1, 3, 5 или 7) количество одновременно появившихся в байте ошибочных битов. Поскольку вероятность одновременного появления трех (а тем более пяти или семи) ошибочных битов в одном и том же байте ничтожно мала, на практике считают, что имеется только один сбойный бит.

Кроме введения контрольного разряда существуют и более развитые способы кодирования, которые не только позволяют установить факт появления ошибок большой кратности, но и обеспечивают их *исправление*. К таким способам относятся коды, принцип построения которых в 1948 г. предложил Р. Хемминг.

Проиллюстрируем общую идею построения кодов Хемминга на простом примере. Пусть имеется исходный 4-битный код 1100_2. Чтобы не только обнаружить, но и исправить вероятную одиночную ошибку, предлагается каждую тройку битов снабдить одним контрольным разрядом. Количество различных троек, которые могут быть выбраны из четырех битов, равно числу сочетаний из четырех по три: $C_4^3 = 4$. Однако теоретический анализ показывает, что можно ограничиться любыми *тремя тройками*. Количество единиц в каждой группе (тройка битов вместе с контрольным разрядом) должно быть нечетным. На рис. 2.16, *а* изображены исходный код и значения контрольных битов **A**, **B** и **C** в каждой выбранной тройке.

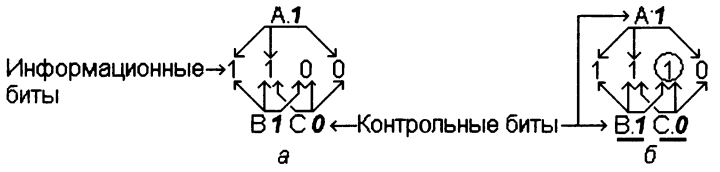


Рис. 2.16. Исправление ошибочного бита: *а* — код без ошибки; *б* — с единичной ошибкой

Если во время выполнения действий один из битов окажется искаженным, то во всех группах, в которые входит ошибочный бит, контроль нечетности покажет наличие ошибки. Следовательно, для определения положения ошибочного бита в коде достаточно выбрать разряд, входящий во все группы, в которых нарушена нечетность. В примере на рис. 2.16 только в группе, контролируемой битом **A**, общее количество единиц нечетное, в то время как значения контрольных битов **B** и **C** (на рисунке эти биты подчеркнуты) сигнализируют о наличии ошибки в их группах. Ошибочный бит должен принадлежать одновременно двум последним группам. Такой бит всего один — это третий по счету бит, отмеченный на рис. 2.16 кружком. После определения положения сбойного бита исправление кода осуществляется его инвертированием.

Теперь рассмотрим предложенный Р. Хеммингом общий порядок построения кодов, обеспечивающий однозначное обнаружение и исправление ошибок в кодах данных. Он включает в себя следующие действия.

1. Контрольные биты вставляются в исходный код и нумеруются совместно с информационными битами слева направо начиная с 1.
2. Контрольные биты располагаются в позициях с номерами $n = 2k$, $k = 0, 1, 2, 3...$ объединенного кода, то есть в позициях с номерами $n = 1, 2, 4, 8, 16...$
3. Для каждого контрольного разряда с номером n весь код делится на группы, состоящие из $2n$ битов.
4. Контрольный бит с номером n контролирует в группе первые n подряд расположенных битов кода (для первой группы, включая контрольный) с пропуском следующих n битов.
5. В каждой группе контрольный бит формируется так, чтобы общее количество единиц в ней было нечетным.

Рис. 2.17. Схема построения кода Хемминга

На рис. 2.17, *а* изображено 32-битное поле, для которого строится код Хемминга. Как показано далее, чтобы иметь возможность исправлять ошибки, в занимающий это поле код необходимо включить шесть контрольных разрядов. На рис. 2.17, *б* контрольные биты закрашены серым цветом, а внизу показана используемая в кодах Хемминга нумерация разрядов поля. В качестве примера формирования контролируемой группы битов на рис. 2.17, *в* выделены разряды, которые контролируются четвертым контрольным битом. Группы битов, закрепленных за первыми четырьмя контрольными разрядами, представлены в табл. 2.8.

Таблица 2.8. Закрепление информационных битов за контрольными

Контрольный бит	Контролируемые биты
1	1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29...
2	2, 3, 6, 7, 10, 11, 14, 15, 18, 19, 22, 23, 26, 27...
4	4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, 28, 29, 30, 31...
8	8, 9, 10, 11, 12, 13, 14, 15, 24, 25, 26, 27, 28, 29, 30, 31...

Для определения ошибочного бита важно знать не какие информационные биты относятся к данному контрольному, а какие контрольные биты закреплены за данным информационным. Анализ показывает, что в общем случае информационный бит с номером b проверяется контрольными битами с номерами b_1, b_2, ..., bk такими, что $b = b_1 + b_2 + ... + bk$. Это значит, что *номер ошибочного бита равен сумме номеров контрольных битов, обнаруживших ошибку.*

Рассмотрим пример построения кода Хемминга и исправления одиночной ошибки для 16-битного кода. Из приведенных далее оценок следует, что для исправления ошибки в таком коде необходимо пять контрольных разрядов. На рис. 2.18, *а* изображен исходный 16-битный код $1111\ 0111\ 1001\ 0111_2$, а на рис. 2.18, *б* — построенный для него код Хемминга, состоящий из 21 бита. Определение значений контрольных битов проиллюстрировано в табл. 2.9: для каждого контрольного бита выписываются все информационные биты, за которые он отвечает, и в них подсчитывается общее количество единиц. Если оно четно, то значение бита должно быть равно 1, в противном случае — 0.

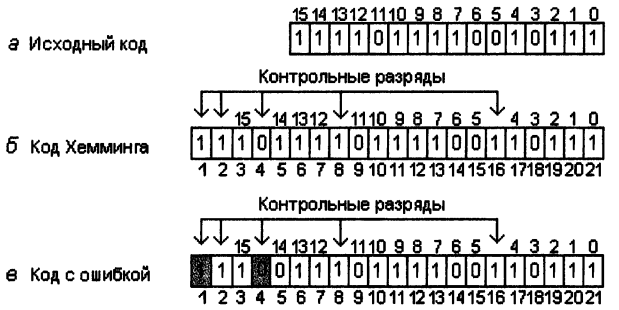


Рис. 2.18. Определение номера ошибочного бита

Таблица 2.9. Формирование значения контрольного бита

Контрольный бит	Контролируемые биты											Количество единиц	Значение бита
1	1	3	5	7	9	11	13	15	17	19	21	8	1
	?	1	1	1	0	1	1	0	1	1	1		
2	2	3	6	7	10	11	14	15	18	19		6	1
	?	1	1	1	1	1	0	0	0	1			
4	4	5	6	7	12	13	14	15	20	21		7	0
	?	1	1	1	1	1	0	0	1	1			
8	8	9	10	11	12	13	14	15				4	1
	?	0	1	1	1	1	0	0					
16	16	17	18	19	20	21						4	1
	?	1	0	1	1	1							

Если в такой код в результате выполнения каких-либо действий попала ошибка, то для определения номера сбойного бита достаточно сложить номера контрольных битов, в группах которых отсутствует нужная четность. Например, на рис. 2.18, *в* ошибочна четность в группах первого и четвертого контрольных битов, — следовательно, ошибочен пятый бит кода.

Количество контрольных разрядов, которые должны быть включены в код, зависит от цели создания кода: предназначен он только для факта обнаружения ошибки или с его помощью должен быть восстановлен исходный код. Кроме того, необходимо принимать во внимание допустимую кратность ошибки. Определение минимально необходимого для заданных условий количества контрольных разрядов базируется на понятии **интервала Хемминга**, а он равен количеству битовых позиций, в которых два кода отличаются друг от друга. Возьмем, например, коды $1000\ 1101_2$ и $0101\ 1001_2$. Они различаются цифрами, находящимися во втором, четвертом, шестом и седьмом разрядах (использован стандартный для байта способ нумерации — слева направо, начиная с нуля). Таким образом, интервал Хемминга d для этих двух кодов равен 4. Интервал Хемминга двух кодов показывает, сколько требуется одновременных однобитовых ошибок, чтобы превратить один код в другой.

Код, в который уже включены контрольные разряды, будем называть **полным кодом Хемминга**. При этом *допустимым* считается такой полный код, у которого правильно заданы значения всех контрольных разрядов. Соответственно код с неправильно заданным значением хотя бы одного контрольного бита считается недопустимым. Обратите внимание: речь идет о построении полного кода, то есть о подборе значений контрольных битов для уже заданного и *неизменного* набора информационных битов.

Если исходный код содержит m информационных разрядов и к нему добавляется k контрольных, то полный код состоит из $m + k$ разрядов. В полном коде из 2^{m+k} возможных кодов *допустимыми* будут только 2^m кодов, так как значения контрольных битов *однозначно* зависят от значений информационных. Таким образом, количество *недопустимых* кодов равно $2^{m+k} - 2^m$.

На рис. 2.19 приведены примеры построения контрольных разрядов (цветом выделены 1-й, 2-й и 4-й разряды полного кода Хемминга) для исходных кодов 0000_2 и 1000_2. Первые семь полных кодов недопустимы, так как указанные в них значения контрольных разрядов не обеспечивают нечетность количества единиц хотя бы в одной из групп (1-, 3-, 5- и 7-й; 2-, 3-, 6- и 7-й; 4-, 5-, 6- и 7-й разряды). Для кода 0000_2 допустим только один полный код 1101000_2, содержащий 1 во всех контрольных разрядах. Далее указано несколько построенных по точно такому же принципу допустимых и недопустимых полных кодов для исходного кода 1000_2.

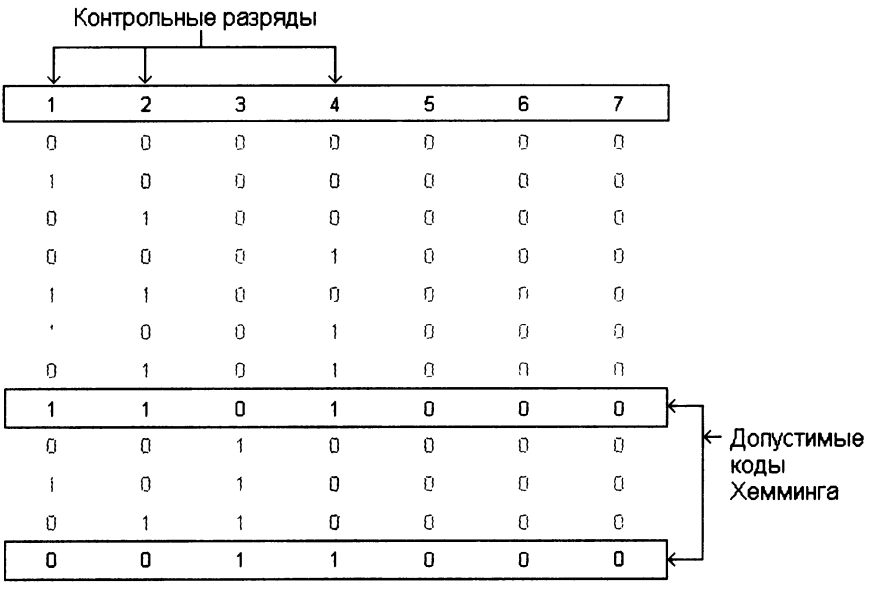


Рис. 2.19. Допустимые коды Хемминга

Зная способ заполнения контрольных разрядов для любого набора информационных битов, можно построить полный список допустимых кодов и найти в этом списке два кода с наименьшим интервалом Хемминга. Этот интервал называется **интервалом Хемминга полного кода**.

Анализ зависимости информационных битов для предыдущего примера (см. рис. 2.18 и табл. 2.9) $m = 16$, $k = 5$, $n = 21$ показывает, что любой информационный бит кода контролируется не менее чем двумя контрольными. Если возникает изменение в единственном информационном бите, то для допустимости нового полного кода должны измениться еще по крайней мере два контрольных бита. Другими словами, интервал Хемминга полного кода в данном примере равен 3.

Получим теперь нижнюю оценку количества контрольных битов, которые необходимы для исправления одиночной ошибки у m-разрядного слова. Каждому из $2m$ допустимых кодов можно сопоставить ровно n недопустимых кодов, которые образуются из допустимого полного кода с *ровно одним* инвертированным битом (рассматриваются только одиночные ошибки). Умножая $2m$ на $n + 1$, получаем общее количество кодов, не содержащих ошибок вообще или же содержащих единственную ошибку. Очевидно, что могут быть полные коды с двумя и большим количеством ошибок в k контрольных разрядах (см. рис. 2.18). Следовательно, число $(n + 1) \cdot 2^m$ должно быть меньше, чем общее количество полных кодов 2^{m+k}: $(n + 1) \cdot 2^m < 2^{m+k}$. Преобразуя это неравенство, получим: $m + k + 1 < 2^k$, откуда $2^k - k > m + 1$. Следовательно, минимальное количество контрольных разрядов k, которые должны быть включены для исправления единичной ошибки в исходный код из m разрядов, должно удовлетворять неравенству $2^k - k > m + 1$.

В табл. 2.10 приведены рассчитанные по этому неравенству количества контрольных битов k для исходного кода длиной m, чтобы полученный при этом полный код Хемминга мог исправлять одиночную ошибку. В этой же таблице приведены длина полного кода Хемминга $m + k$ и относительное увеличение длины исходного кода $L = m/(m + k)$ за счет включения в него контрольных разрядов. Хорошо видно, что уже для кодов с исходной длиной $m \geq 128$ относительное увеличение длины не превышает 6%, что можно считать вполне приемлемой платой за возможность автоматически исправлять ошибки.

Таблица 2.10. Разрядность кодов Хемминга для исправления одиночной ошибки

m	k	$n = m + k$	L
8	4	12	1,5
16	5	21	1,31
32	6	38	1,19
64	7	71	1,11
128	8	136	1,06
256	9	265	1,04
512	10	522	1,02

Более общий анализ показывает, что для обнаружения S ошибок необходим код с полным интервалом $d = S + 1$, а для исправления S ошибок необходим код с полным интервалом $d = 2S + 1$.

Методы построения кодов Хемминга и некоторых других, более мощных способов кодирования широко используются в различных устройствах памяти компьютера. В частности, на этих методах основан аппаратный механизм контроля оперативной памяти **ECC** (Error Checking and Correcting — контроль и исправление ошибок), который является дополнительным средством обеспечения высокой ее помехоустойчивости.

Дополнительный материал по обсуждавшимся в главе вопросам можно найти в изданиях [3], [7], [10], [17], [29].

Контрольные вопросы и упражнения

1. Какие разновидности данных могут быть записаны в памяти компьютера?
2. Что называется форматом данных?
3. Как в компьютерах кодируются текстовые данные?
4. Для чего используются кодовые таблицы? Какие кодовые таблицы вам известны?
5. Как в компьютерах кодируется графическая информация?
6. Дайте определения понятий «пиксел», «растр», «разрешающая способность», «сканирование».
7. Опишите принципы кодирования монохромных изображений.
8. Опишите принципы кодирования цветных изображений.
9. Какие форматы используются для представления числовых данных в компьютерах?
10. Чем отличаются с точки зрения информатики целые числа от вещественных?
11. Какие поля могут быть использованы для хранения целых чисел?
12. Опишите беззнаковое представление формата с фиксированной точкой.
13. Как связан диапазон представления беззнаковых целых чисел с длиной поля?
14. Сформулируйте порядок получения беззнакового кода заданного числа.
15. Получите беззнаковый код во всех возможных полях памяти для следующих чисел: 121_{10}; 469_{10}; 78903_{10}.
16. Сформулируйте порядок определения числа по заданному беззнаковому коду.
17. Определите, каким числам соответствуют следующие беззнаковые коды: $C1_{16}$; B4 06_{16}; 15 A0 02 FE_{16}.
18. Что называется разрядной сеткой?
19. Охарактеризуйте существующие системы знакового кодирования целых чисел.
20. Почему не применяется система кодирования со знаком?
21. Опишите принцип и способы получения дополнительного кода знаковых целых чисел.
22. Как связан диапазон представления знаковых целых чисел с длиной поля?

23. Сформулируйте порядок получения знакового кода для заданного целого числа.
24. Получите знаковый код во всех возможных полях памяти для следующих чисел: -283_{10}; $+199_{10}$; $-35\ 298_{10}$.
25. Сформулируйте порядок определения числа по заданному знаковому коду.
26. Определите, каким числам соответствуют следующие знаковые коды: $E2_{16}$; $AB\ 06_{16}$; $56\ 06_{16}$.
27. Как для заданного кода осуществляется переход от поля с одной длиной к полю с другой длиной?
28. Опишите систему кодирования знакового кодирования со смещением. Где она применяется?
29. Сравните основную и экспоненциальную формы записи вещественных чисел.
30. Что представляют собой нормализованные числа?
31. Опишите особенности выполнения арифметических операций над нормализованными числами.
32. Сформулируйте общие принципы кодирования чисел в формате с плавающей точкой.
33. Как влияет выбор разрядной сетки на диапазон и точность представления вещественных чисел?
34. Что называется переполнением и антипереполнением? Когда они возникают?
35. Что определяет стандарт IEEE 754?
36. Охарактеризуйте особенности стандарта IEEE 754 при записи вещественных чисел в полях длиной 4 байта.
37. Получите для четырехбайтового поля значения максимального и минимального чисел. Какую точность имеют числа в этом поле?
38. Охарактеризуйте особенности стандарта IEEE 754 при записи вещественных чисел в полях длиной 8 байтов.
39. Получите для восьмибайтового поля значения максимального и минимального чисел. Какую точность имеют числа в этом поле?
40. Охарактеризуйте особенности стандарта IEEE 754 при записи вещественных чисел в полях длиной 10 байтов.
41. Получите для десятибайтового поля значения максимального и минимального чисел. Какую точность имеют числа в этом поле?
42. Опишите порядок получения кода заданного числа в формате с плавающей точкой.
43. Получите во всех возможных полях памяти коды в формате с плавающей точкой для чисел $-22{,}5_{10}$; $78{,}35_{10}$; $-1045{,}168_{10}$.
44. Опишите порядок получения числа по его коду в формате с плавающей точкой.
45. Какие числа представлены следующими кодами в формате с плавающей точкой: $E2\ AB\ E4\ 02_{16}$; $C1\ B4\ 00\ 00_{16}$?

46. Как получаются специальные значения стандарта IEEE 754?
47. Охарактеризуйте назначение специальных значений стандарта IEEE 754?
48. Как происходит мягкое исчезновение порядка?
49. В каких ситуациях полезны денормализованные специальные значения? Что дает их использование?
50. В каких ситуациях полезны специальные значения «бесконечность»? Что дает их использование?
51. В каких ситуациях полезны специальные значения «нечисло»? Что дает их использование?
52. Чем отличается компьютерная арифметика от обычной? Приведите примеры отличий.
53. Как производится переход от аналоговой формы сигналов к цифровой?
54. Какие параметры характеризуют цифровую форму звука?
55. Какие мультимедийные форматы данных вам известны?
56. Сформулируйте принцип обезличивания кода.
57. Охарактеризуйте возможные причины появления ошибок в кодах данных при их хранении и передаче в компьютерах.
58. Как обеспечивается надежность кодирования данных в компьютерах?
59. Как обеспечивается надежность на уровне отдельного байта?
60. В чем состоит идея метода построения кодов Р. Хемминга?
61. Опишите порядок построения кода Хемминга.
62. Как определяется положение ошибочного бита в коде Хемминга?
63. Что представляют собой интервалы Хемминга двух кодов и полного кода?
64. Какой код Хемминга нужно построить, чтобы обнаружить и чтобы исправить k ошибок?

Глава 3
Логические основы обработки данных

После выяснения способов представления данных в памяти компьютера необходимо выяснить, как осуществляется их обработка. В данной главе обсуждаются некоторые вопросы, связанные с физическими и логическими основами обработки *дискретных* данных.

3.1. Понятие такта

В выполнении любых действий над данными участвуют несколько устройств компьютера. Совершенно очевидно, что действия, в которых участвуют несколько исполнителей (людей, устройств), должны быть согласованы и синхронизированы друг с другом. В этом смысле можно провести наглядную аналогию между компьютером и оркестром музыкальных инструментов. Чтобы оркестр мог исполнить какую-либо мелодию, музыканты должны действовать строго синхронно, несогласованная игра представляет собой какофонию, а не мелодию. За синхронизацию игры музыкантов в оркестре отвечает дирижер, который в определенном темпе делает взмахи дирижерской палочкой. В компьютере такую же роль играет специальное устройство — **тактовый генератор**, который через равные промежутки времени вырабатывает импульсы синхронизации — **синхроимпульсы**, служащие ориентирами во времени и используемые для координации всеми участвующими в выполнении действия устройствами (рис. 3.1). Длительности вырабатываемых импульсов также одинаковы.

ВНИМАНИЕ

Промежуток времени от начала одного импульса синхронизации до начала следующего за ним импульса называется тактом. Такты обладают равными длительностями с высокой степенью точности.

Выполнение процессором любых действий всегда происходит во время некоторой части такта, а за ней следует пауза, в течение которой ничего не происходит. Наличие такой паузы является принципиальным фактором, обусловленным физическими законами, которые управляют работой процессора. Аналогичным образом обстоят дела при забивании гвоздя молотком. Удары по шляпке гвоздя чередуются с периодами, во время которых молоток поднимается вверх для нанесения удара. Забить гвоздь без таких периодов невозможно. Хотя движение молотка происходит непрерывно, можно считать, что процесс забивания гвоздя в целом распадается на ряд дискретных действий — ударов, происходящих *мгновенно* в моменты соприкосновения молотка и шляпки гвоздя. Точно так же, обсуждая выполнение процессором определенной в программе последовательности действий, можно считать, что любые длящиеся внутри такта действия происходят *мгновенно* в моменты времени t_0, t_1, t_2..., соответствующие границам тактов. Строго говоря, синхроимпульсы, которые считаются границами тактов, также имеют отличную от нуля длительность, поэтому моменты времени t_0, t_1, t_2... следует привязывать к началам тактовых импульсов.

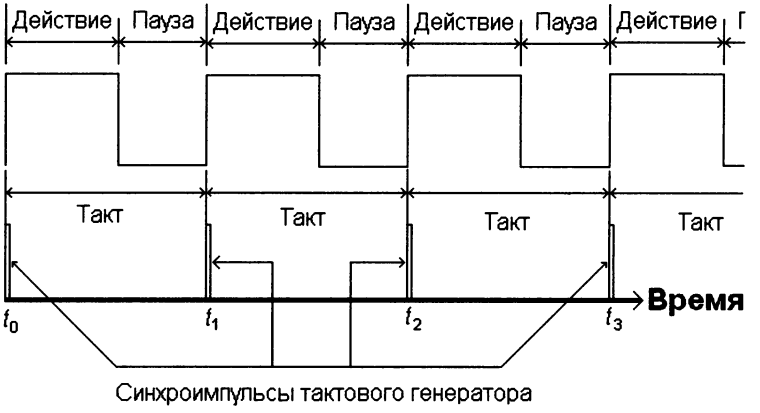


Рис. 3.1. К понятию такта

С понятием такта связана *тактовая частота* — одна из важнейших технических характеристик различных устройств компьютера.

ВНИМАНИЕ

Тактовая частота представляет собой техническую характеристику отдельных устройств компьютера, которая равна количеству тактов, управляющих работой устройства, в единицу времени. Единицей измерения тактовой частоты является герц, равный одному такту в секунду.

Тактовая частота является одним из главных факторов, определяющих скорость обработки данных компьютером. Первые персональные компьютеры работали на частотах 5–10 МГц. В настоящее время процессоры компьютеров работают с тактовыми частотами от сотен мегагерц до 3–4 ГГц. Возможно, в недалеком будущем появятся процессоры с тактовой частотой порядка 10 ГГц. Однако следует

заметить, что, согласно теоретическим оценкам, процессоры, выполненные по современным технологиям, не смогут превзойти частот 30–40 ГГц.

Как известно, программа в общем случае представляет собой некоторый текст, определяющий последовательность действий по обработке данных. Такой текст, так же как и обрабатываемые данные, кодируется в компьютере с помощью двоичного кода. То есть программы, как и данные, представлены в компьютере в дискретном виде.

Для обеспечения задаваемого программой порядка обработки данных в компьютере формируются и используются различные вспомогательные значения и признаки, так или иначе характеризующие результаты обработки и используемые для определения дальнейшего порядка выполнения действий. Так, например, при вычислении любых числовых значений формируется признак нулевого результата. Этот признак имеет значение 1, если в результате вычислений получается 0, и значение 0, если результат ненулевой. Анализ этого признака, например, при вычислении знаменателя дроби, позволяет избежать бессмысленного деления на нуль. Упомянутые значения и признаки, так же как и выполняемая программа в целом, представлены в компьютере двоичным кодом, то есть являются дискретными.

ВНИМАНИЕ

Совокупность, состоящая из двоичного кода выполняемой программы, значений обрабатываемых дискретных данных, а также набора дискретных значений и признаков, которые используются для управления процессом обработки, образует внутреннее состояние компьютера.

С точки зрения введенного понятия внутреннего состояния компьютера, можно считать, что в моменты времени t_0, t_1, t_2... происходит *мгновенный* переход компьютера из одного внутреннего состояния в другое — из состояния, соответствующего значениям обрабатываемых данных и признаков до выполнения действия, в состояние, соответствующее значениям данных и признаков после выполнения этого действия. Таким образом, множество внутренних состояний компьютера также является дискретным. В связи с этим говорят, что компьютер в целом является *дискретным устройством.*

ВНИМАНИЕ

Устройства, обеспечивающие работу с дискретными данными или сигналами, обладающие дискретным множеством внутренних состояний, а также выполняющие действия в дискретные моменты времени, называются дискретными. Компьютер в целом относится к группе дискретных устройств.

3.2. Вентили и комбинационные схемы

Основными, базовыми операциями, которые обязательно должен «уметь» выполнять процессор компьютера над двоичными кодами данных, являются логические операции отрицания, дизъюнкции, конъюнкции, арифметического сложения,

а также сдвига кода. Используемые для реализации этих и других операций устройства принято называть **вентилями** (от нем. Ventil — клапан).

ВНИМАНИЕ

Вентилем называется физическое устройство, реализующее одну из базовых логических операций: отрицание, дизъюнкцию, конъюнкцию, исключающую дизъюнкцию и т. д. Вентили, входящие в состав процессоров компьютера, называют также логическими элементами.

3.2.1. Релейно-контактные вентили

В 1938 г. известный специалист в области теории информации Клод Шеннон предложил использовать для моделирования основных логических операций релейно-контактные электрические схемы. Этот подход был использован в электромеханических релейных вычислительных машинах Z-3 (Германия, Конрад Цузе, 1939), «Марк 2» (США, Говард Айкен, 1947), РВМ-1 (СССР, Н. И. Бессонов, 1951) и в целом ряде других машин.

В качестве примера рассмотрим изображенную на рис. 3.2 реализацию логических операций конъюнкции и дизъюнкции с помощью схем, которые принято называть вентилями «И» и «ИЛИ» соответственно. Логические операнды в этих схемах соответствуют релейно-контактным переключателям, которые на рисунке обозначены как p и q. При этом логические значения 0 и 1 моделируются соответственно разомкнутым и замкнутым состояниями контакта. Результат операции отображается включенной в сеть лампочкой. Если лампочка не горит, результат равен 0; горящая лампочка соответствует результату, равному 1. Иначе говоря, знаки двоичного алфавита 0 и 1 моделируются *отсутствием* или *наличием тока* в цепи соответственно. Заметим: переключатели называются релейными потому, что управление ими обычно осуществляется с помощью электромагнитных реле.

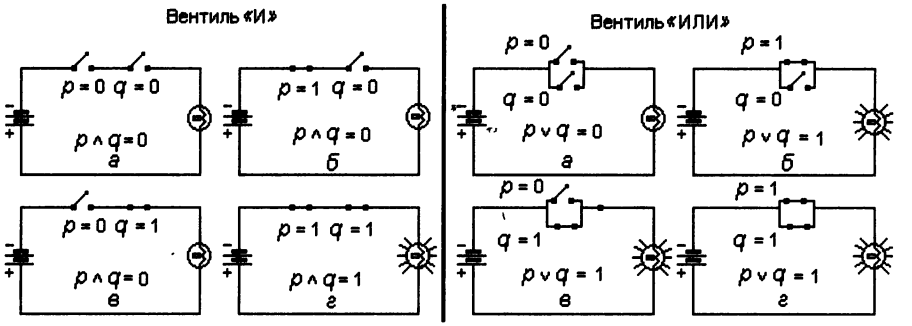


Рис. 3.2. Вентили «И» и «ИЛИ» на базе релейно-контактной схемы

На рис. 3.2, *слева* показаны четыре состояния вентиля «И», которые соответствуют различным строкам таблицы истинности (табл. 3.1) этой операции. Так, в состоянии *а* оба контакта разомкнуты, то есть $p = 0$ и $q = 0$, и, следовательно, ток в цепи не течет — лампочка не горит — $p \wedge q = 0$. Состояния *б* и *в* моделируют

другие строки таблицы истинности операции конъюнкции, в которых результат равен нулю: $1 \wedge 0 = 0$ и $0 \wedge 1 = 0$ соответственно. Ток в цепи течет и лампочка горит только в состоянии *г*, то есть в том случае, когда замкнуты оба контакта, что соответствует строке таблицы $1 \wedge 1 = 1$.

На том же рисунке *справа* изображена схема вентиля «ИЛИ», соответствующего операции дизъюнкции. Лампочка не горит только в состоянии *а*, когда разомкнуты оба контакта, то есть $0 \vee 0=0$. Во всех остальных состояниях лампочка горит, и, следовательно, результат операции равен 1, то есть моделируются строки таблицы истинности $1 \vee 0 = 1$, $0 \vee 1 = 1$ и $1 \vee 1 = 1$. Релейная схема вентиля «НЕ», соответствующего логической операции отрицания, устроена более сложно, поэтому она на рисунке не приводится.

3.2.2. Полупроводниковые вентили

Основным недостатком описанного способа реализации операции над данными является *наличие механических перемещений* контактов релейной схемы, которые требуют значительных временных затрат. Время выполнения операции определяется временем, в течение которого производится замыкание или размыкание контакта. Даже в самых современных релейных устройствах на это требуются по крайней мере сотые доли секунды. Именно это обстоятельство привело к тому, что уже в 1950-е годы релейные вычислительные машины были вытеснены сначала ламповыми, а затем транзисторными компьютерами.

Все современные вентили реализуются на основе полупроводниковых устройств — **транзисторов** (рис. 3.3, *а*) или их аналогов в интегральных схемах, характерное время срабатывания которых в настоящее время приближается к долям наносекунды (1 нс = 10^{-9} с).

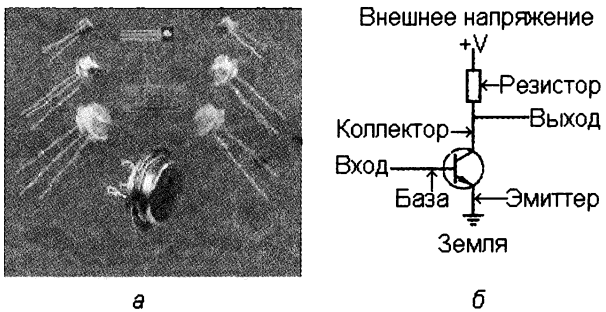


Рис. 3.3. Транзисторы: *а* — внешний вид; *б* — изображение на схемах

В задачу курса не входит изучение физических процессов и явлений, происходящих внутри транзистора. Достаточно понимания его поведения в электрической цепи в нескольких простых ситуациях. Транзистор имеет три контактных вывода, которые принято называть **эмиттером**, **базой** и **коллектором** (рис. 3.3, *б*). Внешнее напряжение (например, +5 В) через резистор подается на коллектор,

а эмиттер заземляется, что создает условия для протекания тока в цепи. Если напряжение на контакте базы отсутствует, то транзистор ведет себя в электрической цепи как резистор с большим внутренним сопротивлением. А если на контакт базы подать напряжение определенной величины, то транзистор ведет себя как проводник с очень маленьким сопротивлением. В первом случае говорят, что транзистор *заперт*, а во втором — что транзистор *открыт*. Контактный вывод базы считается **входом схемы**. Начиная или прекращая подачу напряжения на этот вход, можно управлять поведением транзистора, открывать или закрывать его. **Выходом схемы** считается контактный вывод коллектора, на котором регистрируется результат управляющего воздействия на схему.

В вентильных схемах на базе транзисторов двоичному знаку 0 соответствует низкое напряжение с уровнем от 0 до 1 В, а двоичному знаку 1 — высокое напряжение с уровнем от 2 до 5 В. Могут применяться и другие конкретные значения напряжений, однако в любом случае используются *два четко различимых его уровня*. Подачу на базу транзистора низкого напряжения можно трактовать как поступление на вход схемы бита со значением 0, а подача на базу высокого напряжения соответствует поступлению на вход схемы бита со значением 1. Аналогично регистрация низкого напряжения на коллекторе транзистора может трактоваться как формирование на выходе значения 0, а регистрация на нем высокого напряжения — как формирование на выходе значения 1.

3.2.3. Вентиль «НЕ»

Рассмотрим поведение схемы, изображенной на рис. 3.4, *а*, при различных значениях входного бита p. Пусть на вход схемы подано значение $p = 0$. Тогда транзистор заперт, он ведет себя в цепи как дополнительный резистор с сопротивлением, гораздо большим, чем сопротивление резистора, через который транзистор подключен к источнику питания схемы. В силу того что падение напряжения на участке цепи пропорционально сопротивлению этого участка, напряжение в точке выхода будет мало отличаться от высокого напряжения источника питания. Другими словами, на выходе схемы в этом случае формируется значение 1.

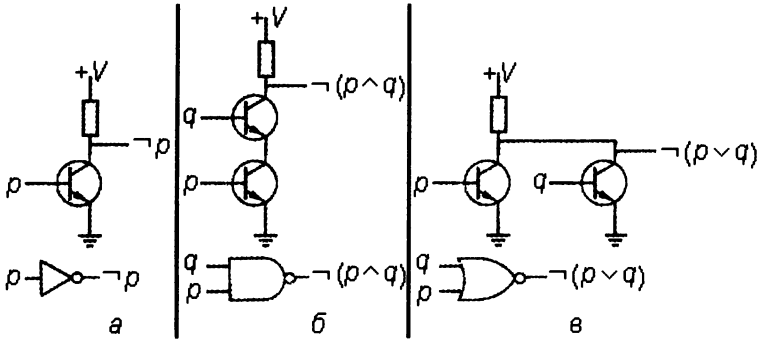


Рис. 3.4. Вентили и их обозначения: *а* — «НЕ»; *б* — «НЕ И»; *в* — «НЕ ИЛИ»

Пусть теперь на вход поступил бит $p = 1$. Тогда транзистор открыт, он ведет себя как проводник с очень маленьким сопротивлением, и все падение напряжения происходит на резисторе[1]. Выход схемы при этом оказывается как бы напрямую соединенным с землей, следовательно, напряжение на нем близко к нулю. Другими словами на выходе схемы формируется значение 0. Таким образом, обсуждаемая схема представляет собой вентиль «НЕ», реализующий одноместную операцию отрицания: при поступившем на вход значении p на выходе схемы формируется значение $\neg p$. Отметим, что время переключения вентиля из одного состояния в другое существенно зависит от физической реализации транзистора. Но в любом случае это переключение для электронных устройств происходит очень быстро, за время, измеряемое микро-, наносекундами или их долями.

3.2.4. Вентили «НЕ И» и «НЕ ИЛИ»

Рассмотрим логику работы схемы, изображенной на рис. 3.4, *б*. Она состоит из двух соединенных последовательно транзисторов. У этой схемы два входа, обозначенных на рисунке буквами p и q, и один выход. Если на входы поступают единичные значения ($p = 1$ и $q = 1$), то оба транзистора открыты, участок цепи с ними имеет очень маленькое сопротивление и, следовательно, как и в вентиле «НЕ», на выходе формируется значение 0. Во всех остальных случаях хотя бы один из транзисторов оказывается запертым и участок цепи с транзисторами обладает высоким сопротивлением, что приводит к формированию на выходе значения 1. Анализируя таблицу истинности работы этой схемы (табл. 3.1, четвертый столбец), приходим к выводу, что она описывается выражением $\neg(p \wedge q)$. Поэтому такая схема называется вентилем «НЕ И». Эта операция известна также под названием «штрих Шеффера». Ее обозначают значком «|»: $\neg(p \wedge \delta 9 q) \equiv p \mid q$.

Таблица 3.1. Таблицы истинностей базовых вентилей

p	q	$p \wedge q$, «И»	$\neg(p \wedge q)$, «НЕ И», штрих Шеффера	$p \vee q$, «ИЛИ»	$p \veebar q$, «Исключающее ИЛИ»	$\neg(p \vee q)$, «НЕ ИЛИ», стрелка Пирса
0	0	0	1	0	0	1
0	1	0	1	1	1	0
1	0	0	1	1	1	0
1	1	1	0	1	0	0

В схеме, изображенной на рис. 3.4, *в*, транзисторы соединены параллельно. Следовательно, участок цепи с транзисторами обладает высоким сопротивлением только в том случае, когда оба транзистора закрыты одновременно. Поэтому ес-

[1] Заметим, что это обстоятельство обусловливает наличие в схеме резистора, через который питание подается на коллектор. Отсутствие резистора в открытом режиме работы транзистора приведет к тому, что все падение напряжения произойдет на обладающем очень маленьким сопротивлением транзисторе и ток большой величины приведет к его перегоранию.

ли на оба входа поступают нулевые значения ($p = 0$ и $q = 0$), на выходе формируется значение 1. Во всех остальных случаях хотя бы один из транзисторов открыт и, следовательно, весь содержащий их параллельное соединение участок цепи обладает маленьким сопротивлением. Это значит, что на выходе схемы формируется значение 0. Анализируя таблицу истинности работы этой схемы (табл. 3.1, седьмой столбец), приходим к выводу, что она описывается выражением $\neg(p \vee q)$. Поэтому такая схема называется вентилем «НЕ ИЛИ». Эта операция известна также под названием «стрелка Пирса». Ее обозначают значком «↓»: $\neg(p \vee q) \equiv p \downarrow q$.

Вентили «НЕ», «НЕ И» и «НЕ ИЛИ», используемые для построения других вентилей и произвольных схем, считаются базовыми, а схемы, которые получаются с помощью всевозможных комбинаций базовых вентилей, принято называть **цифровыми логическими схемами**. Важным частным случаем цифровых схем являются **комбинационные схемы**, в которых значения, получаемые на выходах схемы, зависят только от значений, поступающих на ее входы. Такие схемы классифицируются также как **схемы без памяти**.

Использование в комбинационных схемах стандартных обозначений транзистора и других показанных на рис. 3.4, *б* обязательных элементов (значков земли, питания и т. д.) приводит к излишней громоздкости и затрудненному восприятию схем. Поэтому в комбинационных схемах используются условные обозначения для каждого из базовых вентилей в целом. Условные обозначения трех рассмотренных ранее вентилей приведены в нижней части рис. 3.4 под соответствующими им схемами.

3.2.5. Вентили «И» и «ИЛИ»

Теоретически для задания *любой* логической функции можно обойтись только одной операцией — стрелкой Пирса или штрихом Шеффера. Таким образом, вентили «НЕ И» и «НЕ ИЛИ» могут рассматриваться как *универсальные*, из которых можно составить схему, соответствующую любому логическому выражению. Но получаемые при этом логические выражения и соответствующие им цифровые схемы оказываются чрезвычайно громоздкими и малопонятными. В то же время известно, что логические функции удобно задавать, используя три основные логические операции: отрицание, дизъюнкцию и конъюнкцию. В связи с этим целесообразно использовать в комбинационных схемах вентили, соответствующие этим операциям.

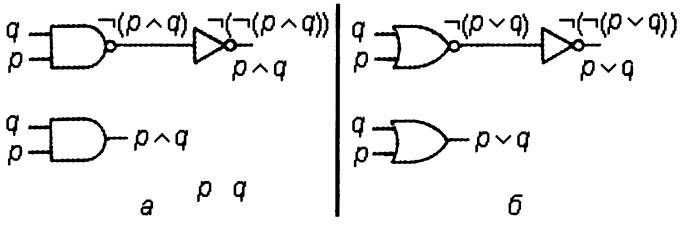


Рис. 3.5. Вентили и их условные обозначения: *а* — «И»; *б* — «ИЛИ»

Способ построения вентилей для операций конъюнкции и дизъюнкции вытекает из очевидных соотношений $\neg(\neg(p \wedge q)) \equiv p \wedge q$ и $\neg(\neg(p \vee q)) \equiv p \vee q$. Следовательно, соединив выходы вентилей «НЕ И» и «НЕ ИЛИ» со входом вентиля «НЕ», получим удобные для построения любых цифровых схем вентили «И» и «ИЛИ» операций конъюнкции и дизъюнкции соответственно. Схемы этих вентилей и их обозначения приведены на рис. 3.5. Вентили «И» и «ИЛИ» также относятся к базовым. Отметим, что для реализации вентиля «НЕ» достаточно одного транзистора, для вентилей «НЕ И» и «НЕ ИЛИ» требуется по два транзистора, а для вентилей «И» и «ИЛИ» необходимо уже по три транзистора на каждую схему.

3.2.6. Построение дизъюнктивной нормальной формы

Наверно, уже понятно, что построение комбинационной схемы, соответствующей устройству, которое обеспечивает выполнение нужных операций, возможно при наличии *предиката*, описывающего логику его работы. Известно, что в общем случае любой предикат, произвольное логическое выражение может быть точно задано с помощью таблицы истинности. Поэтому отправной точкой разработки комбинационных схем обычно является формирование соответствующей логике работы устройства таблицы истинности. Опираясь на таблицу истинности, можно сформировать искомый предикат в виде так называемой дизъюнктивной нормальной формы (ДНФ). Напомним изучаемый в курсе дискретной математики алгоритм построения ДНФ по заданной таблице истинности:

Для каждой строки таблицы истинности составляются *конъюнкты* (элементарные конъюнкции), которые должны содержать *все* аргументы таблицы истинности. При этом аргументы, имеющие в рассматриваемой строке таблицы нулевое значение, включаются в конъюнкт через операцию отрицания.

Выбираются конъюнкты всех строк таблицы, в которых результат равен 1.

Из всех выбранных конъюнктов составляется дизъюнкция, являющаяся искомой дизъюнктивной нормальной формой.

Этот алгоритм опирается на два важных свойства конъюнктов. Во-первых, существует ровно $2n$ различных конъюнктов, в которые фактически входят все n аргументов. А во-вторых, на любом наборе значений аргументов только один из конъюнктов принимает единичное значение, все остальные конъюнкты на нем обращаются в нуль.

Проиллюстрируем построение ДНФ для логической операции (pq) «Исключающее ИЛИ», которая определяется следующим образом: результат операции равен единице, если операнды p и q имеют различные значения, и равен нулю, если они имеют одинаковые значения. Таблица истинности этой операции приведена в шестом столбце табл. 3.1. Построение по описанным ранее правилам конъюнктов для всех строк таблицы дает следующий результат.

p	q	$p \veebar q$	Элементарная конъюнкция
0	0	0	$\neg p \wedge \neg q$
0	1	1	$\neg p \wedge q$
1	0	1	$p \wedge \neg q$
1	1	0	$p \wedge q$

Образуя дизъюнкцию из конъюнктов второй и третьей строк таблицы, в которых результат равен 1, приходим к выводу о том, что выражение $(\neg p \wedge q) \vee (p \wedge \neg q)$ представляет собой ДНФ операции (pq).

3.2.7. Вентиль «Исключающее ИЛИ»

Рассмотренная в предыдущем разделе операция «Исключающее ИЛИ» оказывается полезной во многих случаях построения комбинационных схем. Опираясь на полученную ДНФ этой операции $(p \veebar q) \equiv (\neg p \wedge q) \vee (p \wedge \neg q)$ и используя базовые вентили «НЕ», «И» и «ИЛИ», довольно легко построить соответствующую схему. Для реализации выражения в первой паре скобок выход вентиля «НЕ» следует соединить с одним из входов вентиля «И». Значения p и q подаются на оставшийся свободным вход вентиля «И» и на вход вентиля «НЕ» соответственно. Таким образом, на выходе вентиля «И» получится значение выражения $p \wedge \neg q$. Подключив вентиль «НЕ» к другому входу еще одного вентиля «И» при том же порядке подсоединения p и q, на его выходе получим выражение, находящееся во второй паре скобок, $\neg p \wedge q$. Теперь осталось выходы вентилей «И» соединить со входами вентиля «ИЛИ». Полученная схема вентиля «Исключающее ИЛИ» и его условное обозначение приведены на рис. 3.6, *а*. Для реализации этого вентиля требуется 11 транзисторов.

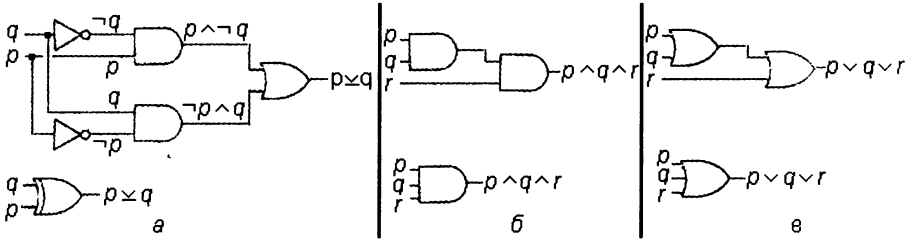


Рис. 3.6. Вентиль «Исключающее ИЛИ» (*а*) и многовходовые вентили (*б* и *в*)

3.2.8. Многовходовые вентили

Стандартные вентили «И», «ИЛИ» являются **двухвходовыми**, то есть они содержат два входа и один выход. В общем случае вентили «И» и «ИЛИ» могут быть **многовходовыми**, то есть содержащими более двух входов. Соответствующие схемы можно получить, увеличивая количество последовательно или параллельно соединяемых транзисторов в схемах, приведенных на рис. 3.5. Многовходовые

вентили можно также построить из стандартных двухвходовых вентилей «И» и «ИЛИ», последовательно соединяя их. На рис. 3.6, *б* и *в* изображены **трехвходовые** варианты таких вентилей. Для реализации n-входового вентиля, очевидно, требуется $n + 1$ транзисторов при использовании первого подхода и $3(n-1)$ — при использовании второго. Подобным образом можно реализовать и многовходовые вентили «НЕ И» и «НЕ ИЛИ»

3.2.9. Комбинационная схема сумматора

Теперь рассмотрим комбинационные схемы, с помощью которых может быть реализовано арифметическое сложение. Анализ алгоритма сложения двоичных кодов показывает, что сложение младших битов и сложение всех остальных битов слагаемых производится по-разному. Различие обусловлено необходимостью учитывать биты переносов для всех битов слагаемых, кроме первого. Комбинационная схема, которая реализует сложение только для двух младших битов слагаемых, называется **полусумматором**, а схема, реализующая сложение для всех остальных битов слагаемых, называется **сумматором**, иногда используется также название **полный сумматор**.

Таблица 3.2. Таблица истинности полусумматора

a	*b*	*a* + *b*	*p*	Σ
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	10	1	0

Введем следующие обозначения. Пусть a и b — участвующие в операции биты слагаемых, Σ — бит результата, а p — бит переноса в следующий разряд. Основываясь на правилах сложения двоичных кодов, получим, что работа полусумматора может быть описана табл. 3.2. Видно, что для бита переноса справедливо соотношение $p = a \wedge b$, а бит суммы Σ получается как результат операции «Исключающее ИЛИ», $\Sigma = a \veebar b$. Схема полусумматора должна иметь два входа, на которые подаются складываемые биты a и b, и два выхода, на которых формируются бит суммы и бит переноса. Эти соображения приводят к изображенной на рис. 3.7 схеме полусумматора. Для ее реализации требуется 14 транзисторов.

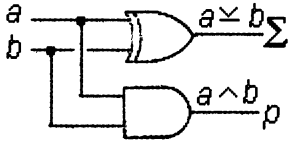


Рис. 3.7. Комбинационная схема полусумматора

При сложении каждой следующей пары битов слагаемых необходимо учитывать бит переноса из предыдущего разряда. Следовательно, эта операция зависит от трех аргументов, а соответствующая комбинационная схема должна иметь три входа. В результате сложения текущей пары битов получается бит текущего разряда суммы и бит переноса в следующий разряд. Поэтому схема должна иметь два выхода. Пусть, как и ранее, a и b обозначают биты слагаемых, а Σ — бит результата. Пусть далее p_{in} — бит переноса из предыдущего разряда, а p_{out} — бит переноса в следующий разряд. Тогда суммирование с учетом переносов можно описать в табл. 3.3.

Таблица 3.3. Таблица истинности полного сумматора

p_{in}	a	b	$a + b + p_{in}$	p_{out}	Конъюнкты для p_{out}	Σ	Конъюнкты для Σ
0	0	0	0	0	—	0	—
0	0	1	1	0	—	1	$\neg p_{in} \wedge \neg a \wedge b$
0	1	0	1	0	—	1	$\neg p_{in} \wedge a \wedge \neg b$
0	1	1	10	1	$\neg p_{in} \wedge a \wedge b$	0	—
1	0	0	1	0	—	1	$p_{in} \wedge \neg a \wedge \neg b$
1	0	1	10	1	$p_{in} \wedge \neg a \wedge b$	0	—
1	1	0	10	1	$p_{in} \wedge a \wedge \neg b$	0	—
1	1	1	11	1	$p_{in} \wedge a \wedge b$	1	$p_{in} \wedge a \wedge b$

Применим технику построения ДНФ для каждого из результирующих битов операции. В табл. 3.3 приведены конъюнкты только для тех строк, которые участвуют в построении соответствующей нормальной формы. Вначале построим выражение для бита Σ (скобки проставлены для упрощения восприятия конъюнктов, из которых образована ДНФ):

$$\Sigma = (\neg p_{in} \wedge \neg a \wedge b) \vee (\neg p_{in} \wedge a \wedge \neg b) \vee (p_{in} \wedge \neg a \wedge \neg b) \vee (p_{in} \wedge a \wedge b).$$

Выполняя элементарные преобразования, получим:

$$\Sigma = \neg p_{in} \wedge ((\neg a \wedge b) \vee (a \wedge \neg b)) \vee p_{in} \wedge ((\neg a \wedge \neg b) \vee (\wedge a \wedge b)).$$

С учетом соотношения $p \veebar q \equiv (p \wedge \neg q) \vee (\neg p \wedge q)$ окончательно находим удобное для реализации в виде комбинационной схемы выражение

$$\Sigma = (\neg p_{in} \wedge (a \veebar b)) \vee (p_{in} \wedge \neg (a \veebar b)) \equiv p_{in} \veebar a \veebar b \veebar p_{in}.$$

Эта схема может быть построена на базе двух последовательно соединенных вентилей «Исключающее ИЛИ». На входы первого вентиля следует подать суммируемые биты, а на входы второго вентиля — выход с первого вентиля и бит переноса.

Построим теперь ДНФ для бита переноса в следующий разряд:

$$p_{out} = (\neg p_{in} \wedge a \wedge b) \vee (p_{in} \wedge \neg a \wedge b) \vee (p_{in} \wedge a \wedge \neg b) \vee (p_{in} \wedge a \wedge b).$$

Группируя первую и последнюю, а также вторую и третью скобки и вынося общие множители, получим удобное для реализации в виде комбинационной схемы выражение

$$p_{out} = \big((a \wedge b) \wedge (\neg p_{in} \vee p_{in})\big) \vee p_{in} \wedge \big((\neg a \wedge b) \vee (a \wedge \neg b)\big) = a \wedge b \wedge p_{in} \wedge (a \veebar b).$$

Значение *ab* для последнего выражения можно снять с выхода первого вентиля «Исключающее ИЛИ».

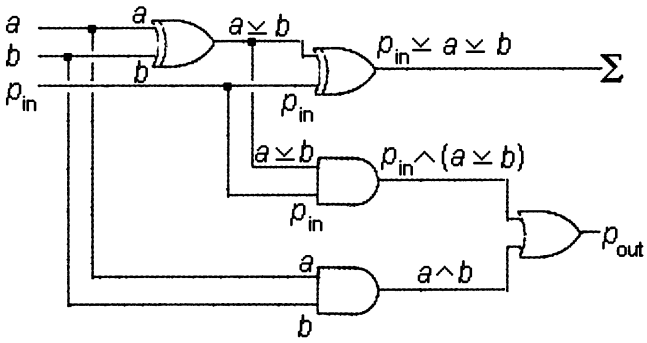


Рис. 3.8. Комбинационная схема полного сумматора

Комбинационная схема полного сумматора приведена на рис. 3.8. Отметим, что для ее реализации требуется 31 транзистор.

3.2.10. Комбинационная схема сдвига

Во время обсуждения умножения в двоичной системе счисления выяснилось, что эта операция, в принципе, сводится к сложению и сдвигу кода. Различают несколько разновидностей сдвига. Сдвиг кода влево означает, что каждый его бит перемещается на соседнюю слева позицию, при этом освободившийся младший (самый правый) разряд поля заполняется нулем, а самый левый бит кода теряется. Про такой бит говорят, что он выталкивается за разрядную сетку. Например, сдвиг кода 00101101_2 влево дает в результате код 01011010_2 (рис. 3.9, *а*). Сдвиг вправо осуществляется в противоположном направлении: каждый бит кода занимает соседний справа разряд, при этом освободившийся старший (самый левый) разряд поля заполняется нулем, а младший бит кода выталкивается за разрядную сетку, теряется. Сдвиг того же самого кода 00101101_2 вправо дает в результате код 00010110_2 (рис. 3.9, *б*). Существуют еще и так называемые *циклические сдвиги кода*, в которых выталкиваемый бит кода не теряется, а записывается в освободившийся слева или справа разряд поля.

Внимательный анализ результатов обычного сдвига двоичного кода показывает, что сдвиг влево эквивалентен умножению на два, а сдвиг вправо эквивалентен целочисленному делению на два. Так, в приведенных ранее примерах коду 00101101_2, рассматриваемому как код в формате с фиксированной точкой, соответствует число 45_{10}. Сдвиг этого кода влево дает в результате код 01011010_2

числа 90_{10}, а сдвиг вправо — код 00010110_2 числа 22_{10}. Для получения корректного результата в случае умножения необходимо, чтобы поле было достаточной для получаемого кода длины, — точнее, чтобы выталкиваемые влево биты не были равны единице.

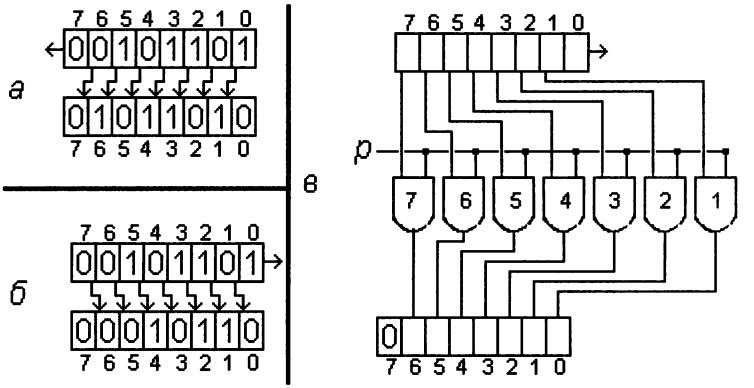


Рис. 3.9. Сдвиг кода: *а* — влево; *б* — вправо; *в* — упрощенная схема сдвига вправо

В рассмотренных ситуациях сдвиг осуществлялся на одну позицию вправо или влево. Имеет смысл рассматривать также сдвиги вправо и влево на несколько позиций, что отвечает умножению или делению на соответствующую степень двойки. Таким образом, сдвиг кода может использоваться не только как вспомогательное действие при реализации общей операции умножения двоичных кодов, но и как самостоятельная операция умножения или целочисленного деления на числа вида $2n$ для целых $n > 0$. Отметим, что операция сдвига выполняется процессорами компьютеров гораздо быстрее, чем общая операция умножения или деления.

На рис. 3.9, *в* представлена упрощенная комбинационная схема, реализующая сдвиг восьмибитного кода вправо на одну позицию. Каждому биту сдвигаемого кода, кроме выталкиваемого младшего бита, соответствует отдельный вентиль «И». На рис. 3.9, *в* эти вентили имеют те же номера, что и соответствующие им биты кода. Каждый бит кода соединен со входом соответствующего ему вентиля, а на второй вход каждого вентиля через единую линию поступает управляющий бит p. Если $p = 1$, то на выходе каждого вентиля дублируется связанный с ним бит кода. Эти выходы соединены с разрядами поля, предназначенного для хранения результата. Обратите внимание: выход каждого вентиля соединен с разрядом поля, номер которого на единицу меньше, чем номер вентиля, что, собственно говоря, и приводит к нужному сдвигу кода. В крайнем слева разряде, для которого нет соответствующего вентиля, автоматически формируется нуль.

Сдвиг кода влево можно организовать с помощью симметричной схемы; в ней выходы вентилей «И» соединены с разрядами поля, номера которых на единицу больше, чем номера соответствующих им вентилей. В более общих схемах оба варианта, которые обеспечивают сдвиги вправо и влево, объединяются в одну

схему с единым управляющим битом, поступающим в две управляющие линии, причем в одну из них через вентиль «НЕ». Таким образом, значение $p = 1$ запускает сдвиг в одну сторону, а значение $p = 0$ обеспечивает сдвиг в другую сторону. В такой схеме сдвига используется $2n$ вентилей «И», где n — количество сдвигаемых разрядов, и один вентиль «НЕ». Следовательно, для ее реализации требуется $6n + 1$ транзисторов.

3.2.11. Компаратор

Еще одной важной операцией, которую, безусловно, должен «уметь» выполнять процессор, является сравнение двух кодов на совпадение. Для ее реализации можно использовать комбинационную схему **компаратора**, восьмибитный вариант которой изображен на рис. 3.10.

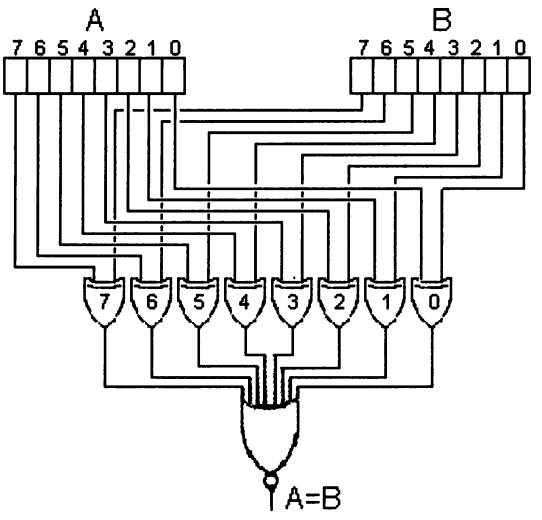


Рис. 3.10. Схема восьмибитного компаратора

Биты сравниваемых кодов A и B, имеющие один и тот же номер, присоединяются ко входам одного и того же вентиля «Исключающее ИЛИ». На рис. 3.10 вентили обозначены номерами, которые совпадают с номерами подключенных к ним битов. Выходы всех вентилей «Исключающее ИЛИ» присоединены к многовходовому вентилю «НЕ ИЛИ». Таким образом, если все биты кодов A и B совпадают, то все вентили «Исключающее ИЛИ» сформируют на своих выходах значение 0, попадающее затем на многовходовый вентиль «НЕ ИЛИ». А если на его входах все нули, то на выходе всей схемы формируется 1. В то же время, если в кодах A и B не совпадает хотя бы одна пара битов, то соответствующий вентиль «Исключающее ИЛИ» выдаст на выходе значение 1. Появление на входах вентиля «НЕ ИЛИ» хотя бы одной единицы приведет к формированию на его выходе значения 0. Таким образом, на выходе компаратора всегда формируется значение

логического выражения A = B. Отметим, что для реализации компаратора, осуществляющего сравнение двух n-битовых кодов, необходимо n вентилей «Исключающее ИЛИ» и один n-входовый вентиль «НЕ И», следовательно, требуется $11n + 2(n - 1) = 13n - 2$ транзисторов.

3.2.12. Декодер и мультиплексор

Выполнение над двоичными кодами таких операций, как дизъюнкция, конъюнкция, сложение и т. д., целесообразно осуществлять с помощью только одной комбинационной схемы, обеспечивающей возможность не только выполнения, но и выбора нужной операции. Выбор одного из нескольких вариантов осуществляется с помощью схемы, которая называется **декодером**.

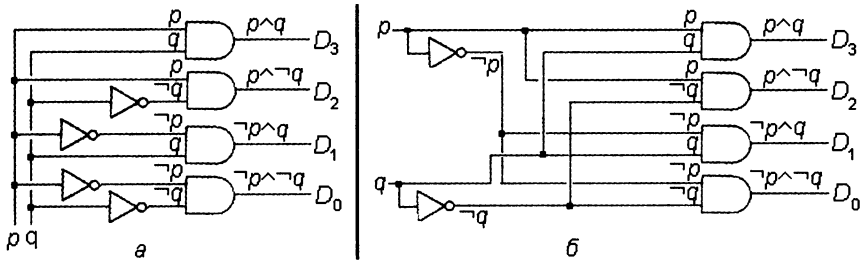


Рис. 3.11. Схема двухвходового декодера

В общем случае в декодере за каждым из рассматриваемых вариантов закрепляется поступающий на входные линии схемы n-разрядный двоичный код. Очевидно, что с его помощью можно закодировать $2n$ различных вариантов. Каждому из них в схеме соответствует отдельный n-входовый вентиль «И», входы которого соединяются с входами схемы либо напрямую, либо через вентиль «НЕ» (рис. 3.11, *а*). Таким образом, все входы отдельно взятого вентиля «И» в совокупности соответствуют конъюнкту, зависящему от n аргументов. Для любого значения поступившего на вход схемы двоичного кода только для одного из вентилей «И» значение конъюнкта оказывается равным 1, и, следовательно, только на его выходе формируется значение 1. На выходах всех остальных вентилей формируется значение 0. Появление на одном из выходов схемы значения 1 трактуется как выбор варианта, который соответствует поступившему на вход управляющему коду.

В качестве примера рассмотрим двухвходовый декодер, изображенный на рис. 3.11, *а*. Этот декодер управляется двухбитовым кодом, обеспечивающим выбор одного из четырех вариантов. В схеме использованы следующие обозначения: p и q — входы, принимающие управляющий код, а D_0–D_3 — выбираемые выходы. Закрепим за выходами схемы управляющие коды: $00_2 \rightarrow D_0$, $01_2 \rightarrow D_1$, $10_2 \rightarrow D_3$, $11_2 \rightarrow D_3$, — и составим конъюнкты, которые определяют подсоединяемые к вентилям входы схемы или их отрицания. В результате за каждым вентилем закрепляется один из четырех возможных конъюнктов. На рис. 3.11 эти конъюнкты

приведены над соответствующими им выходами схемы. При подаче на входы схемы любого из двухбитовых кодов только один из вентилей сформирует на своем выходе 1. Именно этот выход считается выбранным. Пусть, например, $p = 0$ и $q = 1$. Тогда значение 1 имеет только конъюнкт, которому соответствует выход D_1, а на всех остальных выходах формируется 0. Таким образом, декодер по поступившей на входные линии комбинации битов 01_2 выбирает выход D_1.

Для создания n-входового декодера, изображенного на рис. 3.11, *а*, необходимо $2n$ вентилей «И» и в 2 раза меньше вентилей «НЕ», — следовательно, всего требуется $3 \cdot 2^n + 2^{n-1} = 7 \cdot 2^{n-1}$ транзисторов. На рис. 3.11, *б* изображена эквивалентная схема, то есть схема, описываемая точно такой же таблицей истинности, что и схема на рис. 3.11, *а*. Видно, что для реализации правой схемы требуется меньше вентилей «НЕ», чем для реализации левой. Это связано с тем, что отрицания одного и того же входа схемы снимаются с одной и той же линии, а не формируются по отдельности для каждого вентиля «И». Для создания такой схемы необходимо всего n вентилей «НЕ», следовательно, на схему требуется только $3 \cdot 2^n + n$ транзисторов. Приведенный пример показывает, что одной и той же цели можно достичь с помощью различных цифровых схем, обладающих разными требованиями к ресурсам, в частности, к количеству требующихся для их реализации вентилей.

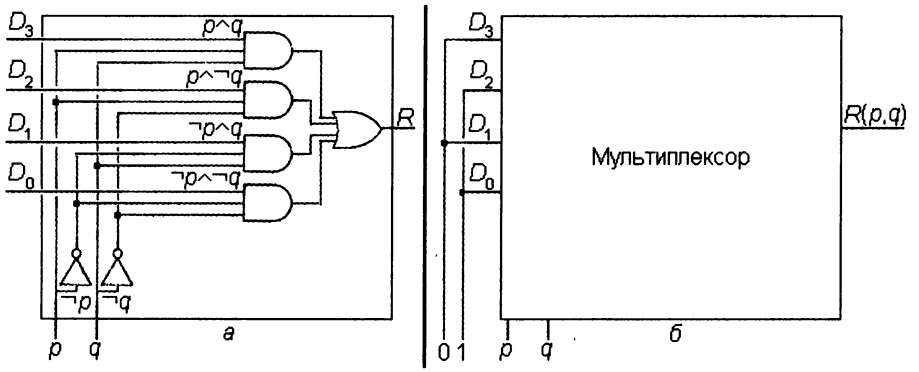


Рис. 3.12. Схемы: *а* — четырехвходового мультиплексора; *б* — реализующая логическую функцию $R(p, q)$

Задача выбора одного из нескольких входных значений и передачи его на выход схемы решается похожим образом с помощью устройства, которое называется **мультиплексором**. В общем случае мультиплексор содержит 2^n основных входных линий, n входных линий управления и один выход. Схема состоит из 2^n многовходовых вентилей «И» и одного также многовходового вентиля «ИЛИ». В качестве примера на рис. 3.12, *а* приведена схема четырехвходового мультиплексора, способного выбрать один из четырех входных битов и передать выбранный бит на единственную выходную линию устройства R. Для выбора нужного варианта в четырехвходовом мультиплексоре используются две линии управления, p и q.

Схема работает следующим образом. Один из входов каждого вентиля «И» считается основным. Он соединяется с одним из выбираемых входов схемы. Каждый из оставшихся n входов вентиля «И» так же, как в декодере, соединяется с одной из управляющих линий схемы либо напрямую, либо через вентиль «НЕ». Таким образом, эти входы вентиля «И» в совокупности соответствуют конъюнкту, зависящему от n аргументов. На рис. 3.12 конъюнкты изображены над соответствующей им линией основного входа вентиля. На управляющие линии схемы подается n-разрядный двоичный код, на котором только один из конъюнктов принимает единичное значение. На выходе соответствующего этому коду вентиля «И» дублируется бит его основной входной линии, а на выходах всех остальных вентилей формируется значение 0. Затем бит из выбранной входной линии через вентиль «ИЛИ» подается на выход мультиплексора.

Обсуждаемая схема может быть использована и в других целях. В частности, на базе мультиплексора можно создать схему, моделирующую любую логическую функцию от n аргументов. Для этого нужно построить таблицу истинности этой функции и подать 1 на все основные входы схемы, соответствующие конъюнктам со значением 1, а на все остальные основные входы подать значение 0. Тогда при поступлении на управляющие входы схемы конкретных значений аргументов на ее выходе получится значение моделируемой логической функции от этих аргументов.

Таблица 3.4. Таблица истинности функции

p	q	$R(p, q)$	Конъюнкт	Вход
0	0	1	$\neg p \wedge \neg q$	D_0
0	1	0	$\neg p \wedge q$	D_1
1	0	1	$p \wedge \neg q$	D_2
1	1	0	$p \wedge q$	D_3

Пусть, например, логическая функция $R(p, q)$ задается таблицей истинности (табл. 3.4). В этой таблице представлены также все конъюнкты и соответствующие им основные входы мультиплексора. Из таблицы следует, что на входы D_0 и D_2 нужно подать значение 1, а на входы D_1 и D_3 — значение 0 (рис. 3.12, *б*). Тогда при поступлении на линии управления p и q значений аргументов на выходе схемы сформируется значение функции $R(p, q)$. Отметим, что на рис. 3.12, *б* специально не изображена внутренняя часть схемы мультиплексора, так как в данном случае нас интересуют только ее входы и выходы, ее внешние линии сопряжения. Такой прием довольно часто используется при анализе сложных схем, которые состоят из нескольких более простых подсхем (рис. 3.14).

Обсудим еще одно важное применение схемы мультиплексора. Она используется для преобразования n битов, *одновременно* передаваемых по разным линиям, в *последовательность* из n битов, передаваемых *друг за другом по одной* линии (рис. 3.13). Такое преобразование приходится выполнять, например, при передаче данных от одного компьютера к другому по линиям связи в компьютерных

сетях, поскольку внутри компьютера биты одного или нескольких байтов обычно передаются между устройствами компьютера одновременно — параллельно, в то время как по внешним линиям связи данные, как правило, передаются последовательно.

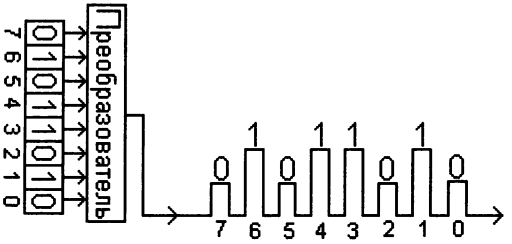


Рис. 3.13. Преобразование параллельного кода байта в последовательный

ВНИМАНИЕ

Одновременная передача по различным линиям нескольких битов называется параллельной передачей, а код, передаваемый таким способом, принято называть параллельной кодом. Если биты кода передают последовательно, друг за другом по одной и той же линии, то такой способ называется последовательной передачей, а передаваемый код называется последовательным кодом.

Для выполнения обсуждаемого преобразования нужно подсоединить к основным входам мультиплексора все линии, по которым одновременно передаются биты. А на его управляющие входы подавать последовательность двоичных кодов, которые осуществляют выбор основных входных линий в желательном порядке. Например, подача на управляющие линии четырехвходового мультиплексора кода 00_2, то есть $p = 0$ и $q = 0$, приведет к выбору основного входа D_0 и передаче находящегося на нем бита на выход схемы. Если немного позднее подать на входы управляющий код 01_2, то на выход попадет бит с основного входа D_1. Последующая подача кодов 10_2 и 11_2 передаст на выход биты сначала со входа D_2, а затем и со входа D_3. Таким образом, параллельно передаваемый код окажется преобразованным в код, передаваемый последовательно. Нужно только своевременно фиксировать или же передавать дальше биты, последовательно попадающие на выход схемы.

3.2.13. Арифметико-логическое устройство

Все функции процессора, связанные с выполнением тех или иных действий над данными, сосредоточены в его внутреннем блоке, который принято называть **арифметико-логическим устройством** (АЛУ).

ВНИМАНИЕ

Часть процессора, выполняющая арифметические, логические и иные операции над данными, называется арифметико-логическим устройством.

Обычно АЛУ, обеспечивающие выполнение действий над n-разрядными данными, состоят из n одинаковых схем, которые выполняют эти действия над двумя битами. Такие схемы называются *одноразрядными* АЛУ. На рис. 3.14 приведена упрощенная схема одноразрядного АЛУ, которое может выполнять логические операции отрицания, конъюнкции, дизъюнкции и арифметического сложения над двумя битами данных.

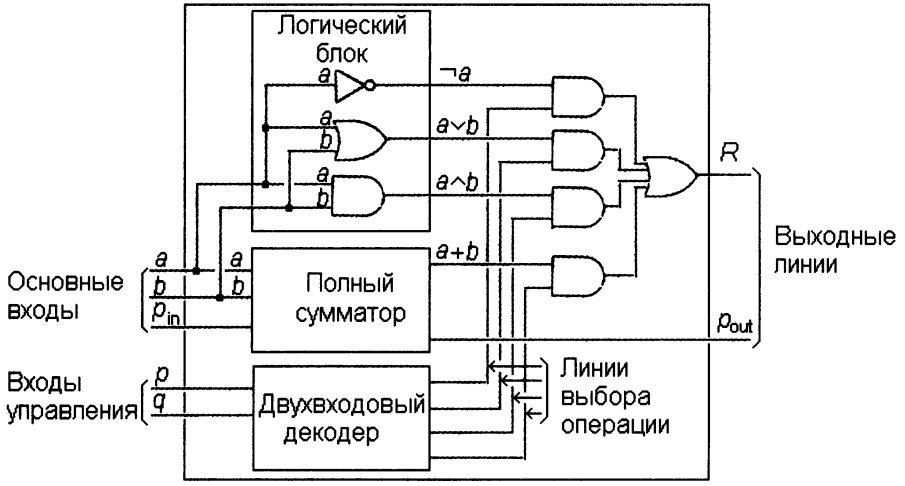


Рис. 3.14. Упрощенная схема одноразрядного АЛУ

В схему по трем основным входным линиям поступают два бита данных, a и b, а также бит переноса из предыдущего разряда p_{in}. Кроме того, в схему поступают два управляющих бита, p и q, значения которых определяют выбор желательной операции. На выходах схема формирует бит результата R и бит переноса в следующий разряд p_{out}. Одноразрядное АЛУ содержит блок выполнения логических операций, отвечающий за операции отрицания, дизъюнкции и конъюнкции, полный сумматор, отвечающий за арифметическое сложение, и декодер, организующий выбор требуемой операции.

На входы логического блока и полного сумматора поступают биты данных a и b, а на вход полного сумматора еще и бит переноса p_{in}. Результаты выполнения операции одновременно формируются на трех выходах вентилей «НЕ», «ИЛИ», «И» логического блока и двух выходах полного сумматора. Бит переноса в следующий разряд p_{out} сразу попадает на выход АЛУ, а результирующие биты операций подаются сначала в подсистему выбора нужного результата. В этой подсистеме для каждой из четырех операций предусмотрен отдельный вентиль «И», на один из входов которого поступает результат этой операции, второй его вход соединен с выходной линей декодера. Декодер, получив на входах некоторую комбинацию управляющих битов p и q, формирует значение 1 на выходе, который соответствует этой комбинации. Таким образом, только тот вентиль «И», который подсоединен к этому выходу декодера, дублирует на своем выходе результат выбранной операции, остальные вентили формируют на выходе нулевое

значение. Чтобы не организовывать несколько отдельных выходных линий из АЛУ, выходы всех вентилей «И» соединены четырехвходовым вентилем «ИЛИ», выход которого является выходом *R* всей схемы АЛУ.

Простой подсчет показывает, что для реализации описанного одноразрядного АЛУ требуется 67 транзисторов, а для аналогичного арифметико-логического устройства, обеспечивающего действия над *n*-разрядными данными, 67*n* транзисторов. Например, для шестнадцатибитного АЛУ нужно 1072 транзистора. Следует иметь в виду, что в учебных целях здесь рассмотрен значительно упрощенный вариант схемы, который отражает только некоторые принципы устройства АЛУ. Реальные АЛУ процессоров могут выполнять значительно большее количество операций, устроены более сложно и, естественно, требуют гораздо большего количества транзисторов.

3.3. Схема памяти на базовых вентилях

Схемы, состоящие из базовых вентилей, применяются не только для создания устройств, выполняющих действия над данными. Они используются также и для реализации одной из разновидностей памяти в компьютере. Но схемы памяти не могут быть отнесены к группе комбинационных схем, так как получаемый на их выходах результат зависит не только от поступивших на вход данных, но и от текущего состояния схемы. *Собственно говоря, эта зависимость и обеспечивает принципиальную возможность запоминания данных.* Схемы, обладающие такими свойствами, относятся к группе последовательных схем. Кроме того, для них используется название **«автоматы с памятью»**.

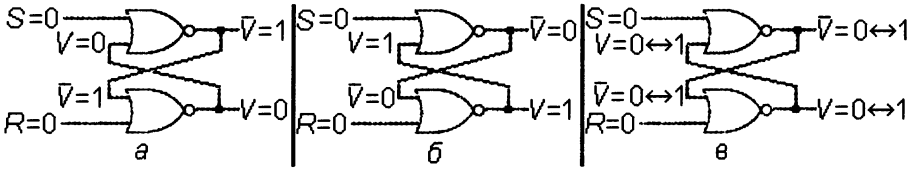


Рис. 3.15. Состояния триггера при нулевых значениях на входах: *а* и *б* — устойчивые; *в* — неустойчивое

Устройство на базовых вентилях, которое можно использовать для хранения одного бита данных, называется **триггером**, или **SR-защелкой**.

ВНИМАНИЕ

Триггером (от trigger — защелка, спусковой крючок) называется устройство, которое может сколь угодно долго находиться в одном из двух состояний устойчивого равновесия и переключается из одного состояния в другое скачком по сигналу или воздействию извне. Триггер может быть реализован, в частности, на электровакуумных лампах накаливания и на полупроводниковых устройствах.

Триггер состоит из двух вентилей «НЕ ИЛИ», выход каждого из которых соединен с одним из входов другого (рис. 3.15). Для управления работой триггера используются свободные входы вентилей, которые обычно называют входом S (от setting — установка) и входом R (от resetting — сброс). От названий входов образовано иногда используемое название этого устройства — SR-защелка. Выходы вентилей V и $\overline{V}$ считаются соответственно *основным* и *дополнительным* выходами всего триггера в целом. Поскольку выходы вентилей V и $\overline{V}$ одновременно являются их же входами, в дальнейшем они называются «входами/выходами» вентилей.

Рассмотрим принцип действия этого устройства. Теоретически возможны 16 различных состояний триггера, соответствующих всем возможным комбинациям значений на входах S и R, а также на входах/выходах Vи $\overline{V}$. Однако, как мы увидим далее, некоторые комбинации значений приводят к возникновению неустойчивых состояний триггера, которые тут же самопроизвольно переходят в устойчивые состояния, способные сохраняться в неизменном виде сколь угодно долго.

Рассмотрим поведение триггера при поступлении на входы значений $S = 0$ и $R = 0$. Вначале допустим, что при этом на входах/выходах уже находятся значения $V = 0$ и $\overline{V} = 1$ (рис. 3.15, *а*). Тогда за один такт работы схемы верхний вентиль сформирует на дополнительном выходе $\overline{V}$ значение $\neg(S \vee V) = 1$, а нижний вентиль сформирует на основном выходе V значение $\neg(R \vee \overline{V}) = 0$. Поскольку эти значения совпадают с исходными, такое состояние триггера является состоянием устойчивого равновесия, или просто устойчивым.

Аналогичная ситуация наблюдается, если в момент поступления на входы триггера S и R нулевых значений на входах/выходах находятся значения $V = 1$ и $\overline{V} = 0$ (рис. 3.15, *б*). Тогда верхний вентиль за такт сформирует на выходе $\overline{V}$ значение $\neg(S \vee V) = 0$, а нижний — на выходе V значение $\neg(R \vee \overline{V}) = 1$. Эти значения также совпадают с исходными, и, следовательно, такое состояние также устойчиво.

Пусть теперь оба входа/выхода находятся в начальном нулевом состоянии $V = 0$ и $\overline{V} = 0$ (рис. 3.15, *в*). Тогда и нижний, и верхний вентили сформируют на выходах V и $\overline{V}$ значения $\neg(R \vee \overline{V}) = 1$ и $\neg(S \vee V) = 1$, которые, вновь поступая на входы вентилей, сформируют на их выходах новые значения $\neg(R \vee \overline{V}) = 0$ и $\neg(S \vee V) = 0$. Очевидно, что такое состояние триггера является неустойчивым, так как он циклически переходит из состояния, при котором $V = 0$ и $\overline{V} = 0$, в состояние, когда $V = 1$ и $\overline{V} = 1$. Теоретически, этот циклический процесс может продолжаться сколь угодно долго. Но для этого необходимо, чтобы одинаковые значения появлялись на выходах строго одновременно. На практике в этой ситуации один из вентилей срабатывает немного быстрее, чем другой, и тогда триггер случайным образом переходит либо в состояние *а*, либо в состояние *б*.

Итак, устойчивыми при $S = 0$ и $R = 0$ являются два состояния, когда на входах/выходах V и $\overline{V}$ находятся различные значения, и неустойчивым — одно состояние, когда на них находятся одинаковые значения. Значение, находящееся на основном выходе триггера V в любом из устойчивых состояний, считается значением, хранящимся в триггере. Таким образом, на рис. 3.15, *а* изображен триггер, сохраняющий значение 0, а на рис. 3.15, *б* — триггер, сохраняющий значение 1.

Рассмотрим поведение триггера при поступлении на его вход S значения 1, которое заменяет обычное его нулевое состояние. Пусть при этом на другом входе R находится нуль, $R = 0$. Допустим, что триггер в это время хранит значение 0. При этом $V = 0$ и $\overline{V} = 1$ (рис. 3.16, *а*). Тогда при срабатывании вентилей на выходах получаются нулевые значения $\neg\left(R \vee \overline{V}\right) = 0$ и $\neg\left(S \vee V\right) = 0$. Следовательно, единица на дополнительном выходе $\overline{V}$ заменяется нулем (рис. 3.16, *б*), после чего срабатывает нижний вентиль $\neg\left(R \vee \overline{V}\right) = 1$ и на основном выходе V формируется 1 (рис. 3.16, *в*). Появление единицы на основном выходе не приведет к дальнейшему изменению состояния триггера, так как $\neg\left(S \vee V\right) = 0$ и значение на дополнительном выходе остается тем же самым. Следовательно, триггер переходит в устойчивое состояние, в котором на основном выходе V находится 1. Другими словами, установка входа S в состояние 1 переводит триггер из состояния 0 в состояние 1. Если же триггер при поступлении 1 на вход S уже хранит единицу, то есть $V = 1$ и $\overline{V} = 0$, то срабатывание его вентилей дает $\neg\left(S \vee V\right) = 0$ и $\neg\left(R \vee \overline{V}\right) = 1$, поэтому его состояние не изменится.

Аналогичным образом можно убедиться в том, что при $S = 1$ и $R = 0$ и исходных состояниях входов/выходов $V = 0$ и $\overline{V} = 0$, а также $V = 1$ и $\overline{V} = 1$ триггер переходит в то же самое устойчивое состояние, при котором $V = 1$ и $\overline{V} = 0$. Следовательно, *при любом исходном состоянии* триггера появление 1 на входе S (при сохранении $R = 0$) переводит его в состояние 1. Можно считать, что эта ситуация эквивалентна записи значения 1 в триггер.

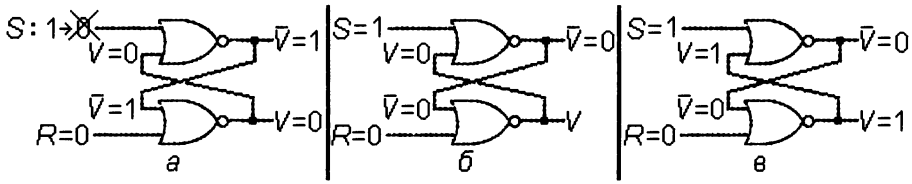


Рис. 3.16. Запись единицы в триггер: *а* — исходное состояние; *б* — промежуточное; *в* — конечное

Пусть теперь значение 1 поступает на вход R, на входе S сохраняется значение 0, а триггер в это время находится в состоянии 0, то есть $V = 0$ и $\overline{V} = 1$. Видно, что текущее состояние триггера эта управляющая комбинация не изменяет, так как срабатывание вентилей дает в результате $\neg\left(R \vee \overline{V}\right) = 0$ и $\neg\left(S \vee V\right) = 1$. Если же в начальном состоянии триггер хранит единицу ($V = 1$ и $\overline{V} = 0$), то на дополнительном выходе формируется $\neg\left(S \vee V\right) = 1$, последующее поступление которой на нижний вентиль дает $\neg\left(R \vee \overline{V}\right) = 0$, то есть триггер переходит в состояние 0. Аналогичным образом можно показать, что при $S = 0$ и $R = 1$ и исходных $V = 0$ и $\overline{V} = 0$, а также $V = 1$ и $\overline{V} = 1$ триггер также переходит в состояние 0. Следовательно, *при любом исходном состоянии триггера* появление 1 на входе R (при сохранении $S = 0$) переводит триггер в состояние 0. Можно считать, что эта ситуация эквивалентна записи 0 в триггер. Итак, триггер запоминает, на каком его входе S или R в последний раз было единичное значение. На основе этого свойства конструируются некоторые реальные схемы памяти компьютера.

Легко убедиться в том, что при поступлении на входы значений $S = 1$ и $R = 1$ триггер может находиться только в одном устойчивом состоянии. При этом оба его выхода равны нулю одновременно, $V = 0$ и $\overline{V} = 0$. Любые другие состояния при этом неустойчивы и случайным образом переходят в указанное устойчивое. В самом деле, пусть, например, в исходном состоянии $V = 0$ и $\overline{V} = 1$, тогда $\neg(R \vee \overline{V}) = 0$ и $\neg(S \vee V) = 0$, и триггер переходит в устойчивое состояние, $V = 0$ и $\overline{V} = 0$. Такой же результат получается и для остальных вариантов исходных значений на входах/выходах V и $\overline{V}$. Поскольку для $S = 1$ и $R = 1$ триггер обладает только одним устойчивым состоянием, эта комбинация для управления его работой не используется.

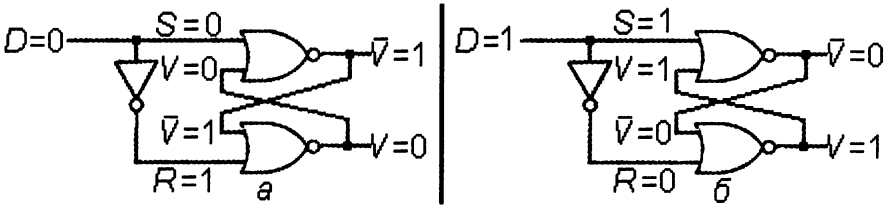


Рис. 3.17. *D*-триггер: *а* — в состоянии 0; *б* — в состоянии 1

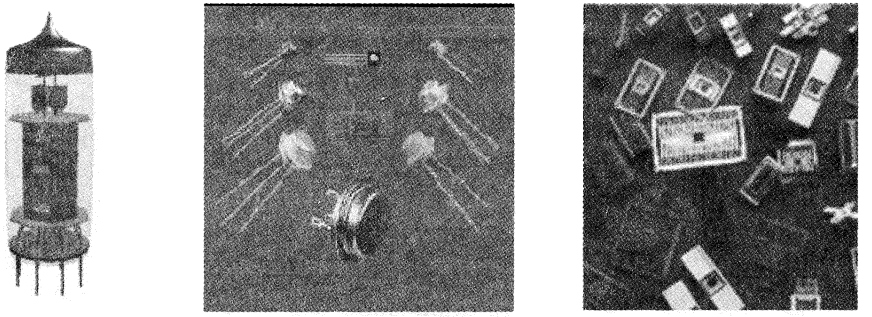
Чтобы избежать одновременного появления на входах единичных значений $S = 1$ и $R = 1$, которое приводит триггер в неиспользуемое состояние, схему триггера немного модифицируют, оставляя у него только один вход, который принято называть входом D (рис. 3.17). Значение с входа D триггера подается на входы S и R верхнего и нижнего вентилей «НЕ ИЛИ», но на один из входов значение попадает через вентиль «НЕ». Таким образом, значения на входах S и R этих вентилей всегда не совпадают. При этом значение на основном выходе триггера V, то есть хранимое в триггере значение всегда совпадает со значением на входе D. Такую схему принято называть D-триггером. На рис. 3.17, *а* показан D-триггер, хранящий значение 0, а на рис. 3.17, *б* — этот же триггер со значением 1.

Итак, описанное устройство позволяет записывать и сохранять сколь угодно долго одну из двоичных цифр, то есть представляет собой одну из возможных физических реализаций элемента памяти компьютера объемом в 1 бит. Чтобы обеспечить удобный способ записи и чтения, а также минимизировать количество необходимых для реализации схемы транзисторов, на практике в качестве бита памяти используются более совершенные схемы D-триггеров, для реализации которых требуется только 6 транзисторов.

3.4. Интегральные схемы

В настоящее время производить, поставлять и использовать для конструирования электронной аппаратуры отдельные вентили не представляется возможным, так как для одного современного не самого мощного компьютера требуются миллионы вентилей. Поэтому для создания вентилей, применяющихся в различных

комбинационных схемах, схемах памяти и любых других устройствах компьютера, а также во многих других электронных устройствах, давно уже не используются транзисторы, изображенные на рис. 3.18, *б*. Все современные аппаратные средства реализуются на **интегральных схемах (микросхемах)**, объединяющих на одной пластинке, которую принято называть **основой**, от нескольких десятков до нескольких сотен миллионов вентилей. Эти вентили функционально полностью эквивалентны рассмотренным ранее, но имеют несравненно меньшие размеры — площадь современных интегральных схем составляет 200–300 мм^2 и менее. Внешний вид интегральных схем приведен на рис. 3.18, *в*.



а *б* *в*

Рис. 3.18. Внешний вид ламп накаливания (*а*), транзисторов (*б*), интегральных схем (*в*)

ВНИМАНИЕ

Интегральная схема (ИС), или микросхема, представляет собой микроминиатюрную электрическую цепь, которая содержит некоторое количество элементов, эквивалентных транзисторам, резисторам и т. д. Вся схема с помощью специальной технологии размещается на очень маленькой кремниевой или какой-либо другой подходящей по свойствам пластинке, точнее, на поверхности или же внутри полупроводникового кристалла.

Отметим, что широкое распространение имеет связанное с очень маленькими геометрическими размерами сленговое название микросхем — **чипы** (от chip — тонкий кусочек).

Микросхема обычно помещается в прямоугольный пластиковый, керамический или иной корпус, снабженный контактными ножками, которые служат входами и выходами схемы. Длина корпусов микросхем колеблется от 0,5 до 5 см.

Пример устройства простейшей интегральной схемы приведен на рис. 3.19. Примерно такими были первые интегральные схемы, появившиеся в конце 50-х гг. XX в. Эта схема состоит из восьми вентилей «НЕ И», собранных в одном корпусе. Для входов и выхода каждого из вентилей на корпусе имеется отдельная контактная ножка. Всего для вентилей схемы требуется 24 контакта. Кроме того, чтобы вентили могли работать, им необходимы питание и заземление. Для этого на корпусе имеется еще две контактные ножки. Вентили такой интегральной

схемы могут произвольным образом (либо поодиночке, либо все вместе) использоваться в конструируемом устройстве.

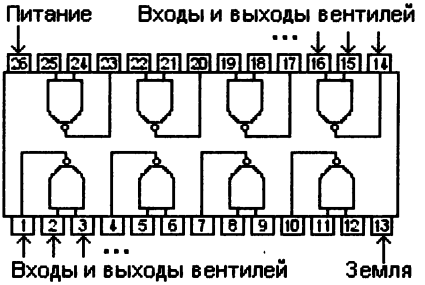


Рис. 3.19. Пример простейшей интегральной схемы, состоящей из восьми вентилей «НЕ И»

Понятно, что обеспечить отдельными контактами на корпусе каждый вентиль интегральной схемы, состоящей даже из нескольких тысяч (не говоря уже о миллионах) вентилей, — задача бессмысленная, да и принципиально неразрешимая. Поэтому используются специальные технологические и схемные решения, которые обеспечивают размещение на площади 2–3 см^2 десятков и сотен миллионов вентилей, использующих при этом всего несколько сотен контактов.

Интегральные схемы характеризуются **степенью интеграции**. Она численно равна количеству транзисторов (иногда вентилей), из которых состоит схема. Степень интеграции используется для деления микросхем на четыре группы:

- малые интегральные схемы (МИС) — до 10^2 транзисторов;
- средние интегральные схемы (СИС) — от 10^2 до 10^3 транзисторов;
- большие интегральные схемы (БИС)— от 10^3 до 10^5 транзисторов;
- сверхбольшие интегральные схемы (СБИС) — свыше 10^5 транзисторов.

Кроме того, часто используемыми параметрами микросхем являются линейный размер вентиля и уже упоминавшаяся ранее площадь микросхемы (не путать с площадью корпуса, в который она запаяна!). Современные вентили имеют линейные размеры, измеряемые микрометрами (1 мкм = 10^{-6} м) и нанометрами (1 нм = 10^{-9} м).

Интегральные схемы обычно специально создаются для решения конкретных задач, например таких, как хранение или обработка данных, и название микросхемы включает ее специализацию: микросхема памяти или микросхема процессора. Микросхемы процессоров обычно называют еще более кратко — **микропроцессор**.

ВНИМАНИЕ

Микропроцессором называется интегральная схема, обеспечивающая выполнение всех функций процессора компьютера.

Возможно также создание многоцелевых, или универсальных, интегральных схем, которые могут приспосабливаться конструкторами для решения различных задач.

В принципе, схема, изображенная на рис. 3.19, несмотря на ее простоту, может быть отнесена к группе универсальных схем.

Дополнительный материал по обсуждавшимся в главе вопросам можно найти в изданиях [4], [29], [34], [35].

Контрольные вопросы и упражнения

1. Какую роль играет тактовый генератор в работе компьютера?
1. Дайте определения понятий «такт» и «тактовая частота».
2. На что влияет значение тактовой частоты?
3. Какими тактовыми частотами характеризуются первые компьютеры? Какими тактовыми частотами обладают современные компьютеры? Чем ограничен рост тактовых частот компьютеров?
4. Какие устройства относятся к дискретным?
5. Что представляет собой вентиль?
6. Охарактеризуйте достоинства и недостатки релейно-контактных вентилей.
7. Изобразите схемы релейно-контактных вентилей «И» и «ИЛИ» и опишите их возможные состояния.
8. Нарисуйте схему транзистора и опишите его закрытое и открытое состояния.
9. Нарисуйте схему и условное обозначение базовых вентилей «НЕ», «НЕ И», «НЕ ИЛИ», «И», «ИЛИ», а также опишите их работу.
10. Опишите алгоритм построения дизъюнктивной нормальной формы.
11. Нарисуйте схему и условное обозначение вентиля «Исключающее ИЛИ», опишите его работу.
12. Каким образом можно реализовать многовходовые вентили?
13. Постройте предикаты, описывающие работу полусумматора, и нарисуйте его комбинационную схему.
14. Постройте предикаты, описывающие работу полного сумматора, и нарисуйте его комбинационную схему.
15. Нарисуйте комбинационную схему сдвига кода и опишите ее действие.
16. Постройте комбинационные схемы для следующих предикатов: $(\neg a \vee b) \wedge \wedge (a \vee c \vee d)$, $\neg a \wedge \neg b \vee \neg (a \wedge c) \wedge \neg a \wedge b$, $\neg (\neg a \wedge b \vee \neg b \wedge c) \vee a \wedge \neg d$.
17. Для чего используется компаратор? Нарисуйте его комбинационную схему и опишите принцип действия.
18. Для чего используется декодер? Нарисуйте его комбинационную схему и опишите принцип действия.
19. Для чего используется мультиплексор? Нарисуйте его комбинационную схему и опишите принцип действия.

20. Для предикатов из упр. 16 нарисуйте схему, основанную на использовании мультиплексора.
21. Дайте определения понятий «последовательный код» и «параллельный код».
22. Опишите основанный на использовании мультиплексора способ преобразования параллельного кода в последовательный.
23. Нарисуйте схему одноразрядного арифметико-логического устройства и опишите принцип его действия.
24. Чем отличаются комбинационные схемы от схем с памятью?
25. Что представляет собой триггер? Нарисуйте его схему.
26. Сколько возможных состояний имеет триггер? Какие состояния триггера считаются устойчивыми, а какие неустойчивыми?
27. Опишите поведение триггера при записи в него единицы для всех возможных исходных состояний.
28. Опишите поведение триггера при записи в него нуля для всех возможных исходных состояний.
29. Опишите особенности *D*-триггера. Зачем понадобилось изменять основную схему триггера?
30. Что представляет собой интегральная схема? Нарисуйте простой вариант универсальной интегральной схемы.
31. Дайте классификацию интегральных схем по степени интеграции.
32. Что представляет собой микропроцессор?

Глава 4
Архитектура компьютера на базе процессора i8086

Различные модели вычислительных машин существенно отличаются друг от друга по своему устройству. Для более подробного начального изучения выбрана архитектура персонального компьютера IBM PC (от International Business Machines Personal Computer) на базе микропроцессора i8086 (или просто 8086). Буква i в маркировке отображает название производителя процессора — фирмы Intel (от Integrated Electronics). В литературе для этого процессора иногда встречается обозначение iAPX8086, где iAPX образовано от Intel Advanced Processor Architecture — продвинутая процессорная архитектура фирмы Intel.

Модель микропроцессора i8086 является в некотором смысле базовой для всех остальных моделей IBM PC. По сути дела, с появлением этой модели микропроцессоров персональные компьютеры начали стремительными темпами завоевывать и изменять мир.

4.1. Основные устройства компьютера

Для обсуждения принципов работы компьютера необходимо более подробно обсудить некоторые особенности его важнейших устройств: оперативной памяти, процессора, шины и т. д.

4.1.1. Оперативная память

Память в компьютере, обеспечивающая одну из важнейших его функций — *хранение* данных и программ, представлена группой устройств, которые отличаются друг от друга физической реализацией, способом использования, объемом, стои-

мостью и некоторыми другими характеристиками. Из всех видов памяти компьютера наиболее важное значение имеет **оперативная память** (ОП). Компьютер принципиально не в состоянии работать, если в нем отсутствует оперативная память, в то время как отсутствие других видов памяти только снижает эффективность его работы, но возможность выполнять программы и обрабатывать данные остается.

Уровень оперативной памяти компьютера подобен кратковременной памяти человека. Когда человек сосредоточен на выполнении какого-либо дела — играет на музыкальном инструменте, управляет автомобилем, — он хорошо помнит все детали, подробности текущей ситуации, а также план выполняемой работы. После перехода к другой деятельности все это забывается, но в памяти возникают другой план и другие подробности.

ВНИМАНИЕ

Оперативная память представляет собой устройство компьютера, которое предназначено для хранения программ, находящихся на стадии выполнения, а также обрабатываемых этими программами данных.

Согласно определению, в оперативной памяти на стадии выполнения может одновременно находиться несколько программ. Кроме того, в ней могут находиться как обрабатываемые, так и уже обработанные или же ожидающие обработки данные для всех выполняющихся процессором программ.

Важнейшим отличительным свойством оперативной памяти является то, что процессор компьютера имеет непосредственный доступ ко всей информации, которая в ней находится. Программы, находящиеся в оперативной памяти, могут быть выполнены процессором, а данные, находящиеся в ней, могут быть по этим программам обработаны. Кстати, именно потому что процессор может оперировать данными и программами, этот вид памяти называется оперативной памятью.

Оперативная память *энергозависима*. Это значит, что при отключении электропитания компьютера все программы и данные, которые в этот момент находятся в оперативной памяти, *безвозвратно теряются*. Наряду с другими устройствами компьютера память характеризуется важным показателем — **быстродействием**, которое в дальнейшем рассматривается более подробно. Пока будем считать, что эта характеристика определяется временем, которое требуется для получения из памяти запрошенных данных, — чем меньше промежуток времени от момента запроса данных до момента их получения из памяти, тем больше скорость памяти.

Ранее было выяснено, что элементарными устройствами любого вида памяти компьютера являются биты, которые предназначены для хранения одной двоичной цифры. Биты оперативной памяти могут быть реализованы различными цифровыми логическими схемами. Одна из них — это рассмотренный ранее *D*-триггер. Другие возможные реализации битов оперативной памяти рассматриваются в 5.3 и 7.1.

Следующим уровнем в организации памяти являются байты. С логической точки зрения, байт состоит из восьми бит, каждый из которых независимо от остальных может содержать любую двоичную цифру, — следовательно, байт служит для хранения восьмиразрядного двоичного кода.

Все байты в пределах оперативной памяти пронумерованы идущими подряд целыми числами, при этом номер начального байта всегда считается равным нулю. Номер байта, заданный в обсуждаемой в дальнейшем форме (см. 4.1.3), принято называть его **адресом**. Нумеруются байты памяти в двоичной системе счисления, но, как правило, номера записываются в шестнадцатеричном виде (рис. 4.1, *сверху*).

Рис. 4.1. Нумерация байтов и стандартные поля оперативной памяти

Содержимое любого байта памяти может задаваться и обрабатываться независимым от остальных байтов образом. После получения адреса любого байта оперативной памяти процессор компьютера может прочитать код, который в нем записан, или записать в этот байт какой-либо другой код. В то же время процессор не может получить доступ к отдельному биту байта. Для этого он должен сначала обратиться к байту, который содержит нужный бит.

Из-за того что есть возможность напрямую обращаться к любому байту, оперативную память называют еще **прямоадресуемой памятью**, или **памятью с прямым доступом**, и используют для нее обозначение RAM (от Random Access Memory — память произвольного доступа). Кроме того, для оперативной памяти применяются и другие названия и обозначения: **оперативное запоминающее устройство** (ОЗУ), **основная оперативная память** (ООП), а иногда просто **основная память** (ОП).

В 1.3 введено понятие поля, которое представляет собой рассматриваемую как единое целое группу смежных байтов памяти. Байт поля с наименьшим номером считается младшим, а байт поля с наибольшим номером считается его старшим байтом. Поле памяти характеризуется адресом и длиной. Адресом поля считается номер его младшего байта, а длиной поля считается количество байтов, из которых оно состоит (рис. 4.2). Байты внутри поля нумеруются слева направо. Напомним, что биты внутри полей памяти и ее отдельных байтов принято нумеровать справа налево, также начиная с нуля. На рис. 4.2, *внизу*, в увеличенном виде показаны пример содержимого и нумерация байтов и битов поля длиной 4 байта.

Принято различать стандартные и нестандартные поля. Нестандартные поля могут иметь произвольную длину и любой адрес, в то время как на длину и адрес стандартных полей накладываются определенные ограничения, которые зависят от архитектуры компьютера. Нестандартные поля используются в основном для хранения текстовых данных, а стандартные — для хранения числовых данных различных форматов.

Рис. 4.2. Поле оперативной памяти

Различают следующие типы стандартных полей: **полуслово**, **слово**, **двойное слово**, **учетверенное слово**. Базовым стандартным полем считается слово, длина которого является одной из основных характеристик архитектуры компьютера. Длины всех остальных разновидностей полей очевидным образом связаны с их названиями. В архитектуре процессора i8086 слово памяти имеет длину 2 байта, то есть 16 бит. Соответственно полуслово имеет длину 1 байт, двойное слово — 4 байта, а учетверенное слово — 8 байтов. Существуют и другие варианты длины слова памяти. Укажем, например, на еще один распространенный в настоящее время вариант, в котором слово состоит из четырех байтов. Тогда полуслово имеет длину 2 байта, а двойное и учетверенное слова — 8 и 16 байтов соответственно.

Адреса стандартных полей не могут быть произвольными, они обязательно должны быть кратными длине поля: адрес слова должен без остатка делиться на 2, адрес двойного слова — на 4, адрес учетверенного слова — на 8. Адрес состоящего из одного байта полуслова может быть любым. Слова памяти всегда начинаются с четных адресов: 0, 2, 4, 6..., двойные слова — с адресов 0, 4, 8, C, 10..., а учетверенные слова — с адресов 0, 8, 10, 18... Примеры стандартных полей приведены на рис. 4.1. Отметим, что изображенное на рис. 4.2 поле не относится к стандартным полям. Несмотря на то что оно имеет длину 4 байта, это поле нельзя считать двойным словом, так как его адрес не кратен четырем. На указанном рисунке изображено нестандартное поле длиной 4 байта.

4.1.2. Процессор

Следующая основная функция компьютера — обработка данных, осуществляемая по заранее заданной человеком программе. Как мы уже выяснили, эта функция выполняется устройством, которое называется процессором, иногда **центральным процессором** (ЦП), или CPU (от Central Processing Unit — центральный обрабатывающий блок, устройство), а в персональных компьютерах еще и микропроцессором.

Рассматривая комбинационные схемы, мы выяснили, каким образом могут быть в принципе реализованы устройства, обеспечивающие выполнение базовых операций над двоичными данными, таких как сравнение, сложение, сдвиг и т. д. Напомним, что рассматривались упрощенные варианты этих устройств. Реальные

процессоры содержат их аналоги в той или иной усовершенствованной форме. Для повышения удобства составления программ, а также для управления последовательностью выполнения действий возможности процессоров по обработке данных существенно расширены. Реальные процессоры могут выполнять сотни различных действий, не входящих в обсуждавшийся базовый набор операций. Все множество действий, которые могут быть выполнены процессором, называется его **системой команд**. Отметим, что процессоры разных моделей компьютеров обладают различными системами команд. Поэтому система команд процессора фактически определяет модель компьютера. Можно также утверждать, что модель компьютера определяет его систему команд.

ВНИМАНИЕ

Системой команд процессора называется набор действий над данными, которые могут быть им выполнены. Система команд процессора взаимно-однозначно связана с его моделью.

Машинной командой (или машинной инструкцией) называется закодированное в двоичном алфавите указание процессору на выполнение отдельного действия, принадлежащего к его системе команд.

Конкретная последовательность машинных команд, которая обеспечивает необходимую обработку данных, называется программой, записанной на уровне машинного языка, или просто машинной программой.

Итак, машинная программа представляет собой некоторую последовательность двоичных кодов, поэтому ее часто называют также **программным кодом**. Заметим, что команды программы расположены в оперативной памяти не обязательно подряд на сплошном ее участке. Запись программы в виде двоичных кодов — это единственный способ задания программы, в котором она непосредственно воспринимается, «понимается» и выполняется процессором компьютера. Все остальные способы записи программ являются промежуточными или вспомогательными.

Процессоры компьютеров характеризуются рядом параметров. Основными считаются **тактовая частота** и **длина машинного слова**. Понятие тактовой частоты рассмотрено в 3.1. Отметим, что тактовая частота различных процессоров, даже одной и той же модели, может изменяться в широких пределах. Так, например, тактовая частота процессора i8086 в зависимости от его модификации колеблется в пределах от 5 до 10 МГц.

Процессор выполняет каждую машинную команду программы за определенное количество тактов. Скажем, существуют процессоры, в которых операция сложения двух чисел в формате с фиксированной точкой выполняется за два такта. А операция деления может занять и 25 тактов такого процессора. Таким образом, можно сделать следующий вывод: чем выше тактовая частота, тем быстрее работает компьютер. Другими словами, скорость работы, или **быстродействие** компьютера, которое может характеризоваться количеством машинных команд, выполняемых компьютером за одну секунду, связано с его тактовой частотой. Одновременно мы выяснили, что быстродействие зависит и от выполняющейся

программы, от того, какие команды — сложения или, скажем, деления — в ней преобладают. Если взять программу, в которой имеются только команды типа сложения, выполняющиеся за два такта, тогда быстродействие процессора с тактовой частотой 500 МГц оценивается в 250 млн. машинных команд в секунду. Если же в программе преобладают действия типа деления, то быстродействие оказывается равным только 20 млн. команд в секунду. Поэтому быстродействие определяют на специальных тестовых программах, которые дают представление как о наивысшей возможной скорости вычислений, так и о предполагаемой средней скорости обработки данных.

Вычислительная мощность компьютера определяется также количеством байтов, которые могут быть одновременно обработаны процессором. Чем больше это количество, тем больше данных в единицу времени может быть обработано.

ВНИМАНИЕ

Машинным словом называется наибольшая группа байтов, которая может быть обработана процессором за один машинный такт. Количество байтов в машинном слове называется длиной машинного слова.

Разные модели машин имеют машинные слова, содержащие различное количество байтов. Первые персональные компьютеры могли за такт переслать или обработать всего один байт, это значит, что машинное слово состояло из одного байта.

В процессоре i8086 машинное слово состоит из двух байтов, то есть длина машинного слова совпадает с длиной слова памяти. Другими типичными для настоящего времени длинами машинных слов являются 4 и 8 байтов. Длина машинного слова обычно выступает в качестве основной характеристики архитектуры компьютера. Так, если машинное слово состоит из одного байта, то есть из 8 битов, то говорят: «восьмибитная архитектура», «восьмибитный компьютер», а если из двух байтов, то есть из 16 битов, то говорят: «шестнадцатибитная архитектура», «шестнадцатибитный компьютер» и т. д.

4.1.3. Шина

Во время выполнения программы процессор постоянно обращается к оперативной памяти. Он выбирает из оперативной памяти команды программы и обрабатываемые данные, а также записывает в память результаты их обработки. Для передачи всей этой информации процессор и оперативная память соединяются между собой набором проводов. По каждому проводу передается только один бит данных, бит адреса или управляющий сигнал. Контроль над правильностью передачи информации по проводам обеспечивают специальные электронные схемы.

ВНИМАНИЕ

Комплекс, состоящий из набора проводов, по каждому из которых передается один бит, и электронных схем, обеспечивающих правильную передачу данных внутри компьютера, называют системной шиной (магистралью), общей шиной или просто шиной.

Количество проводов в шине называется ее **разрядностью**, а провода, из которых она состоит, называются **линиями шины**. Легко заметить, что по линиям шины осуществляется параллельная передача кода, о которой шла речь в 3.2.11.

Для соединения с различными устройствами в шине предусмотрены разъемы стандартизированных форм и размеров. Разъемы шины делятся на **сокеты** (от socket — углубление, гнездо), *слоты* (от slot — узкая прорезь, паз) и **порты** (от port — вырезы в бортах судов для входа и выхода пассажиров, погрузки и разгрузки грузов). Сокеты, обычно имеющие квадратную форму, и узкие щелевидные слоты предназначены для крепления микросхем процессоров и памяти, а порты служат для подсоединения внешних устройств компьютера.

На рис. 4.3 в условном виде изображены фрагмент двадцатиразрядной шины и ее разъем. На контактах разъема показан пример группы битов, одновременно передаваемых по линиям шины.

Рис. 4.3. Фрагмент двадцатиразрядной шины

Из памяти в процессор по шине передаются очередные команды выполняющейся программы и данные, которые должны быть обработаны. Из процессора в память на хранение передаются уже обработанные данные, а также адреса новых запрашиваемых данных. Кроме того, по шине передается необходимая управляющая информация, обеспечивающая правильность передачи. Для передачи разных видов информации в шине используются различные группы проводов, различные линии. Линии шины, по которым передается необходимая управляющая информация, образуют **шину управления**. Часть шины, которая служит для передачи адресов полей или байтов оперативной памяти, называется **адресной шиной**. А когда говорят о линиях шины, отвечающих за передачу содержимого этих байтов, используют название **шина данных**.

Разрядность шины данных тесно связана с длиной машинного слова и часто бывает ей равна. В компьютере с процессором i8086 используется шестнадцатиразрядная шина данных, то есть ее разрядность совпадает с длиной машинного слова и длиной слова памяти.

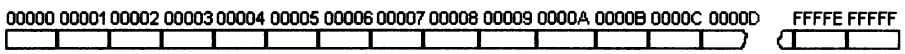


Рис. 4.4. Адреса байтов памяти в компьютере с двадцатиразрядной адресной шиной

Каждая комбинация битов на линиях адресной шины (см. рис. 4.3) считается адресом байта или поля оперативной памяти. Этот адрес всегда записывается двоичными или шестнадцатеричными цифрами, количество которых определяется разрядностью адресной шины. Так, например, при *двадцатиразрядной* адресной шине, которая используется в компьютере на базе процессора i8086, адрес любого байта оперативной памяти задается *ровно двадцатью* двоичными или же *пятью шестнадцатеричными* цифрами (рис. 4.4). Именно в форме записи и состоит упоминавшееся ранее отличие номера байта от его адреса.

ВНИМАНИЕ

Физический адрес, или просто адрес байта оперативной памяти — это его порядковый номер, заданный соответствующим разрядности адресной шины количеством двоичных или шестнадцатеричных цифр.

Так как байт оперативной памяти может рассматриваться как частный случай поля, в дальнейшем изложении вместо оборота «адрес байта или поля памяти» как его сокращенный эквивалент применяется выражение «адрес поля памяти».

Используя физическую аналогию, можно сказать, что нулевой байт оперативной памяти является началом абсолютной системы отсчета адресов; при этом физические адреса байтов в этой системе отсчета являются абсолютными адресами.

Разрядность адресной шины играет важную роль в архитектуре компьютера, так как она определяет *максимально возможный объем* оперативной памяти компьютера. Так, при двадцатиразрядной адресной шине существует всего 2^{20} различных комбинаций — различных адресов, состоящих из двадцати двоичных цифр. Следовательно, для обсуждаемой адресной шины возможна адресация максимум 2^{20} байт, что составляет 1 048 576 байт, или 1 Мбайт.

ВНИМАНИЕ

Множество адресов оперативной памяти компьютера, находящихся в интервале от нулевого до максимально возможного адреса, который определяется разрядностью адресной шины, называется линейным адресным пространством или просто адресным пространством. Объем адресного пространства определяется максимально возможным адресом и является важнейшей характеристикой компьютера.

Фактический объем оперативной памяти равен увеличенному на единицу максимальному номеру байтов, из которых она состоит. Этот объем, в отличие от объема адресного пространства, у разных экземпляров одной и той же модели компьютеров может быть различным.

Шина связывает между собой не только процессор и оперативную память: фактически, все устройства компьютера — диски, клавиатура (от лат. clavis — ключ,

то есть совокупность клавишей в различных устройствах), дисплей и т. д. — так или иначе принимают и передают данные через шину (рис. 4.5). Именно поэтому ее называют общей шиной.

Рис. 4.5. Упрощенная схема компьютера с шинной архитектурой

Рассмотрим в упрощенном варианте порядок обмена данными между устройствами компьютера по общей шине. За каждым из подключенных к общей шине устройств компьютера закрепляется индивидуальный код устройства, который обеспечивает возможность точного задания получателя информации. При передаче данных, например, в оперативную память устройство-отправитель, например процессор, формирует на линиях адресной шины адрес поля памяти, в который должна быть произведена запись. Кроме того, на линиях шины данных формируется записываемый в это поле код, а на линиях управления — признак записи и коды устройств получателя и отправителя (в данном случае — оперативной памяти и процессора). Эту информацию одновременно получают все устройства компьютера, подключенные к общей шине, но воспринимает ее только одно устройство, код которого размещен на линиях шины управления как код устройства-получателя. В данном примере оперативная память, определив, что код на шине данных предназначен для нее, записывает его в поле, адрес которого находится на адресной шине. Аналогичным образом происходит передача данных по общей шине и между другими устройствами, скажем, из оперативной памяти в процессор, на жесткие диски или дисплей компьютера и т. д. Различие только в наборе и трактовке кодов на линиях шины управления и адресной шины.

Поскольку к общей шине обращаются все устройства компьютера, которые, вообще говоря, могут выполнять свои действия независимо и одновременно, вполне вероятна ситуация, когда несколько устройств в одно и то же время обратятся к шине с запросом на передачу по ней данных. Для разрешения таких конфликтных ситуаций предусмотрена специальная микросхема, которую называют **арбитром шины**, а выбор устройства, получающего доступ к шине, называется **арбитражем шины**.

Микросхема-арбитр по специальному алгоритму определяет, какое из обратившихся устройств имеет право на передачу, а какие устройства будут ждать, пока шина освободится. Для этого присоединенным к общей шине устройствам при-

писываются специальные приоритеты. Устройство с наивысшим приоритетом получает разрешение на доступ к шине, а все остальные ставятся в очередь на обслуживание. Обычно устройства ввода/вывода имеют более высокий приоритет перед процессором, так как во многих случаях из-за конструктивных особенностей (например, механических перемещений, вращения и т. д.) они не могут ждать, пока шина освободится.

Компьютеры, в которых для внутренней передачи данных используются шины, относят к компьютерам с **шинной архитектурой**. В современных компьютерах шины делятся на **внутренние** и **внешние**. Внутренняя шина находится внутри процессора и используется для передачи кодов между его внутренними частями, а внешние шины соединяют устройства вне центрального процессора. Кроме того, в составе вычислительной системы предусмотрено несколько специализированных шин, которые соединяют друг с другом оперативную память и процессор, процессор и дисплей и т. д. Такая схема обеспечивает существенное повышение мощности и эффективности работы компьютера. Специализированные шины, входящие в состав современного компьютера, отличаются друг от друга разрядностью и еще одной важнейшей характеристикой — **скоростью обмена**, которую у шин называют еще и **производительностью** или **пропускной способностью**. Скорость обмена шины определяется как количество передаваемых по шине байтов в единицу времени. От разрядности и скорости обмена зависят тип и количество устройств компьютера, которые могут быть подсоединены к той или иной шине.

4.1.4. Внешние устройства компьютера

Оперативная память и процессор образуют центральную группу устройств компьютера. Кроме этих устройств и соединяющей их шины в состав компьютера входит группа внешних устройств (ВУ), в которой выделяют **внешнюю память** (ВП), или **внешние запоминающие устройства** (ВЗУ), и **устройства ввода/вывода**, которые иногда называют **устройствами I/O** (от input/output — ввод/вывод). В шинной архитектуре все внешние устройства, так же как процессор и оперативная память, подсоединены к общей шине (см. рис. 4.5), по которой организуется обмен данными между всеми входящими в компьютер устройствами.

Внешние запоминающие устройства образуют отдельный уровень памяти компьютера, который является аналогом вспомогательных средств, используемых человеком для долговременного хранения важных сведений, — записных книжек, всевозможных справочников, фотографий, звукозаписей, кинопленок, видеозаписей и т. д. Такие носители информации естественно считать внешними по отношению к «внутренней» памяти, «находящейся» в голове человека.

ВНИМАНИЕ

Внешней памятью называется группа устройств, которые предназначены для долговременного хранения больших массивов информации — программ и данных. В отличие от оперативной памяти, устройства внешней памяти не могут быть непосредственно доступны процессору компьютера.

Так как процессор не имеет непосредственного доступа к внешней памяти, находящиеся в ней программы *не могут* выполняться, а данные *не могут* быть каким-либо образом обработаны. В этом и состоит самое главное функциональное отличие внешней памяти от оперативной. Внешнюю память компьютера можно представлять себе как значительный по объему *информационный склад*, где программы и данные могут храниться годами до тех пор, пока они не потребуются.

Во внешней памяти программы и данные хранятся, так сказать, «в нерабочем состоянии», а в оперативной они находятся во время выполнения (и только во время выполнения) программ. Чтобы выполнить какую бы то ни было программу, ее сначала нужно «взять со склада» — найти на внешнем устройстве и перенести в оперативную память, где она и сможет выполняться. Аналогично, чтобы обработать данные, физически находящиеся во внешней памяти, их нужно сначала перенести в оперативную память.

ВНИМАНИЕ

Перенос программы из внешней памяти в оперативную называется загрузкой, а инициирование (начало) ее выполнения называют запуском программы или передачей управления этой программе.

Важнейшей особенностью внешней памяти является ее **энергонезависимость**. Это означает, что информация хранится в ней независимо от того, включено или выключено электропитание компьютера. Кроме того, внешняя память имеет значительно большие объемы по сравнению с оперативной. Скорость передачи данных при обмене с внешними запоминающими устройствами значительно меньше, чем у оперативной памяти, но стоимость оперативной памяти значительно выше, чем стоимость внешней.

В настоящее время в качестве внешней памяти в основном используются гибкие магнитные, жесткие магнитные и оптические диски. Можно упомянуть также и магнитные ленты, хотя их использование в общем-то устарело. Гибкие магнитные и оптические диски являются сменными носителями информации. Это значит, что за счет смены дисков в дисководах на такие носители можно записать неограниченно много информации, хотя объем каждого отдельно взятого диска может быть и не очень большим. Кроме сменных дисковых устройств в состав персональных компьютеров включается один или несколько постоянных, несъемных дисков, которые называют **жесткими** (Hard Disk — жесткий диск), или **винчестерами**. Получить доступ к жесткому диску, не разобрав корпус компьютера, обычно невозможно.

К группе устройств ввода/вывода относятся: клавиатура, используемая для набора данных и выполнения простейших операций по управлению работой компьютера, манипулятор «мышь», используемый для управления, а также ввода простейшей графической информации, дисплей, служащий для отображения текущей ситуации и полученных результатов вычислений, принтер, модем, обеспечивающий связь между компьютерами по станадартным телефонным линиям, и целый ряд других устройств. Устройства этой группы служат для ввода

(передачи данных от человека к основным устройствам компьютера — оперативной памяти, процессору) или вывода (передачи данных от основных устройств компьютера к человеку). Другие устройства этой группы обеспечивают обмен информацией между двумя и более различными компьютерами или между компьютерами и какими-либо другими измерительными устройствами или исполнительными механизмами.

В состав компьютера кроме перечисленных наиболее часто используемых устройств могут быть включены и некоторые другие устройства для обмена данными. Более подробно внешние устройства компьютера рассматриваются во второй части учебника.

4.1.5. Программная модель оперативной памяти

В процессе разработки и написания программ программист абстрагируется, отвлекается от конкретных физических и технических особенностей, от деталей реализации битов и байтов оперативной памяти. Он представляет себе память с точки зрения ее эффективного использования при написании программ, принимая во внимание только необходимые для этого свойства памяти. Другими словами, программист использует **программную модель** оперативной памяти.

Оперативная память как последовательность машинных слов

Оперативную память компьютера естественно рассматривать как упорядоченную в порядке возрастания адресов совокупность байтов. Однако любое обращение к памяти со стороны процессора по ряду технических соображений, которые обеспечивают ускорение передачи данных, осуществляется не по произвольным адресам байтов, а только по адресам слов памяти. Это означает, что процессор может за одно обращение к оперативной памяти прочитать содержимое слова с адресом, например, 00004_{16}. Но процессор не может обратиться в память ни к байту с адресом, например, 00005_{16}, ни к нестандартному полю с этим же адресом (см. рис. 4.2 и 4.1), ни к любому другому байту или полю с нечетным адресом. Если все-таки требуется выполнить обращение по нечетному адресу, то процессор осуществит его за несколько шагов. Например, при чтении сначала будут выполнены обращения по четным адресам соседних слов, а затем из выбранных слов будут отобраны требуемые байты. Так, для выборки содержимого двухбайтового поля с адресом 00005_{16} (байты 00005_{16} и 00006_{16}) процессор прочитает слова с адресами 00004_{16} (байты 00004_{16} и 00005_{16}) и 00006_{16} (байты 00006_{16} и 00007_{16}), а затем отберет из их содержимого код, соответствующий байтам 00005_{16} и 00006_{16}.

Эта процедура, требующая нескольких обращений в оперативную память, естественно, занимает больше времени, чем одиночные обращения к ее словам. Поэтому для размещения данных в оперативной памяти, которое, заметим, целиком зависит от разработчика программы, предпочитают выбирать поля с четными адресами. Такой прием называется **выравниванием данных на границах слов**. В связи с этим

программисты, работающие на уровне машинных команд, часто рассматривают оперативную память как упорядоченную последовательность слов памяти с четными адресами (рис. 4.6), а не как последовательность ее байтов.

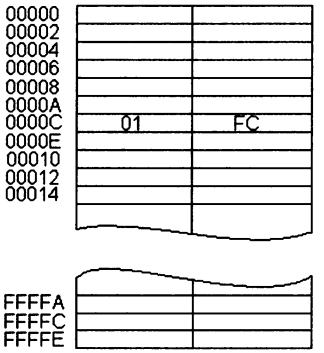


Рис. 4.6. Программная модель оперативной памяти

Кроме выравнивания на границах слов в различных случаях размещения данных и программных конструкций в оперативной памяти используются выравнивания на границах двойных слов, учетверенных слов и так называемых параграфов. **Параграфом** называется участок оперативной памяти, который всегда начинается с адреса, кратного шестнадцати, — 0_{16}, 10_{16}, 20_{16}... и т. д. (см. рис. 4.1). Отметим, что адреса параграфов заканчиваются одним шестнадцатеричным или же четырьмя двоичными нулями.

Принцип обратной записи

Разработчики вычислительной техники часто используют общий принцип, по которому более важная информация должна размещаться в начале сообщения, а ее уточнение, различные детали — в конце. В соответствии с этим принципом размещаются, например, элементы кода числа в формате с плавающей точкой: в начальном разряде поля находится самый важный элемент — код знака, затем располагается код порядка и на последнем месте находится наименее информативный код мантиссы, причем сначала размещаются старшие биты мантиссы, а в конце — ее младшие биты. Действие указанного принципа можно наблюдать и в повседневной жизни, достаточно, например, прислушаться к объявлениям о прибытии поездов на вокзалах.

Применение обсуждаемого принципа к размещению кодов данных в полях памяти с учетом ряда технических соображений реализуется в виде правила, по которому старшие байты полей, то есть байты с большими номерами, должны содержать более важную информацию.

ВНИМАНИЕ

В процессорах Intel старшие (начальные) биты кода занимают старшие байты поля, а младшие (завершающие) биты кода занимают соответственно младшие байты поля.

С точки зрения этого правила обратим внимание на способы нумерации байтов и битов внутри полей памяти (см. рис. 4.2). Байты внутри полей, так же как в оперативной памяти в целом, нумеруются слева направо, что, вообще говоря, соответствует естественному способу чтения текстов и выполнения нумерации в большинстве национальных языков. В то же время биты внутри отдельных байтов и полей памяти нумеруются справа налево. Необходимость в использовании такого порядка нумерации битов объясняется действием закона разложения по основанию системы счисления, в соответствии с которым в позиционных системах определяются значения чисел. В нумерации справа налево номер любого бита совпадает с номером разряда соответствующей цифры кода, что обеспечивает эффективное выполнение арифметических операций над кодами целых чисел.

Порядок нумерации байтов внутри поля, очевидно, противоречит правилу размещения старших и младших битов кода. Рассматривая, например, содержимое поля на рис. 4.7, *а* как код числа в формате с плавающей точкой, увидим, что несущий самую важную информацию знаковый бит попадает в нулевой байт поля, а наименее информативный код мантиссы находится в первом, втором и третьем байтах. Причем младшие цифры кода мантиссы находятся в старшем, третьем байте, а ее старшие цифры — в первом. Таким образом, нужно отказываться либо от выбранных способов нумерации, либо от обсуждаемого принципа. Разработчики процессоров Intel разрешили это противоречие, введя обратный порядок записи кодов, в соответствии с которым старшие цифры кода всегда записываются в старшие байты поля.

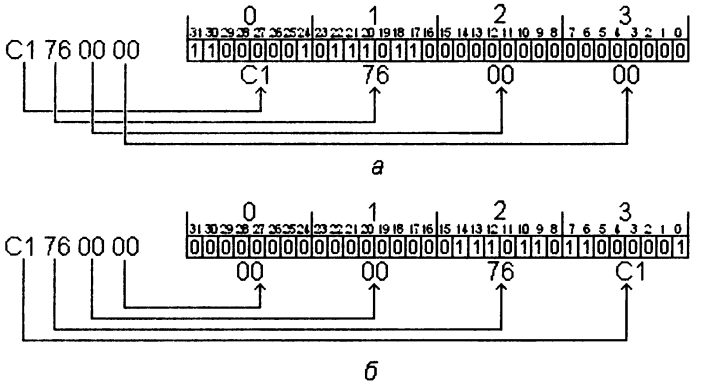


Рис. 4.7. Порядок записи кода в поле: *а* — прямой; *б* — обратный

На рис. 4.7, *а* и *б* показаны два способа записи кода в поле: прямой и обратный. Использование прямого способа сохраняет естественный порядок следования битов кода в поле, но нарушает обсуждаемый принцип — старшие разряды кода попадают в младшие байты поля, в то время как обратный порядок записи удовлетворяет этому принципу.

Все необходимые действия по поддержанию обратного порядка записи кодов в поля памяти выполняются аппаратурой компьютера автоматически. Поэтому

программисту в большинстве случаев о нем можно не беспокоиться. На практике об обратном порядке записи необходимо помнить, когда приходится анализировать содержимое различных полей или участков оперативной памяти. Рассмотрим пример, изображенный на рис. 4.6. Если не вспомнить о принятом способе записи кодов, можно прийти к естественному, но *ошибочному* выводу о том, что поле с адресом $0000C_{16}$ содержит код $01FC_{16}$, который в знаковом представлении формата с фиксированной точкой соответствует числу $+508_{10}$. Учитывая обратный порядок записи, приходим к выводу о том, что данное слово памяти фактически содержит код $FC01_{16}$, который в той же интерпретации соответствует числу -1023_{10}. Сложности возникают также в случае передачи информации в компьютер, процессор которого использует другой порядок записи, и в некоторых других ситуациях.

Сегментация оперативной памяти

Естественное на первый взгляд представление об оперативной памяти как о совокупности байтов, каждый из которых в равной мере доступен любой из одновременно выполняющихся программ, приводит к ряду проблем, которые в основном сводятся *к невозможности организации одновременного выполнения программ, осуществляющих обращение для записи к одним и тем же физическим адресам оперативной памяти.*

Рассмотрим суть этих проблем более подробно. Легко понять, что запись по одним и тем же физическим адресам оперативной памяти двух различных программных кодов или используемых ими кодов данных приведет к неверной работе или даже уничтожению по крайней мере одной из таких программ. Поэтому в программах следует избегать прямого задания физических адресов полей памяти. Этот же вывод подтверждается и следующими рассуждениями. Известно, что программы поступают в компьютер на выполнение в случайном порядке. Операционная система автоматически выделяет для записи и хранения кодов каждой программы и ее данных подходящий по размерам свободный участок оперативной памяти. Вследствие случайности поступления программ на выполнение гарантировать выделение программе того участка памяти, который содержит зафиксированные в программе физические адреса, невозможно, так как он может быть занят другой программой, поступившей на выполнение ранее.

Выявленные негативные факторы влекут за собой требование о наличии у программ свойства **перемещаемости**, которое заключается в том, что программа не должна зависеть от выделенного ей участка оперативной памяти. Другими словами, на любом подходящем по размерам участке памяти программа должна выполняться правильно, и, следовательно, ее можно как бы перемещать по адресам оперативной памяти, загружая ее в разные участки памяти в зависимости от необходимости.

Итак, возникает задача выбора такого способа указания полей памяти в программах, который не использует физические адреса и избавляет от обсуждавшихся проблем. Механизм, обеспечивающий программам нужные свойства перемещаемости и защиты от взаимного разрушения во время их одновременного нахождения

в оперативной памяти, представляет собой **сегментацию** оперативной памяти. Центральным понятием этого механизма является **сегмент памяти**, который отличается использованием автономной, независимой от общей адресацией полей, памяти.

ВНИМАНИЕ

Сегментом называется сплошной участок оперативной памяти с независимой адресацией байтов, входящих в сегмент.

Каждый сегмент оперативной памяти, так же как и поле памяти, характеризуется адресом начала сегмента (или просто адресом сегмента) S и длиной сегмента L. Заметим, что **адрес сегмента** S является абсолютным адресом его начального байта, а адрес $S + L$ является абсолютным адресом последнего байта сегмента. Внимательный анализ показывает, что понятия «поле памяти» и «сегмент памяти» очень близки, если не тождественны. Различие между этими двумя понятиями проявляется в характерных длинах участков, а также в способе их использования. Поля имеют длину от единиц до сотен байтов, а сегменты — десятки и сотни тысяч байтов и более. Поле, как правило, использует для хранения кода одного числа или строки символов, в то время как сегмент служит для хранения кода программы, а также большого количества кодов данных, используемых программой.

В каждом сегменте памяти кроме общей, абсолютной системы адресации вводится своя собственная, независимая от общей относительная система отсчета адресов байтов, принадлежащих сегменту. Адреса D таких байтов отсчитываются относительно начального байта сегмента, поэтому их называют **внутрисегментными смещениями**, или просто **смещениями**. Итак, начальный байт сегмента является началом относительной системы отсчета, а внутрисегментное смещение D некоторого байта является относительным адресом, который отсчитывается от этого начала. Отсюда ясно, что абсолютный адрес A любого находящегося внутри сегмента байта равен сумме абсолютного адреса начала сегмента S и относительного адреса, представляющего собой внутрисегментное смещение D (рис. 4.8, *а*). Такой способ задания адресов называется **сегментной адресацией**.

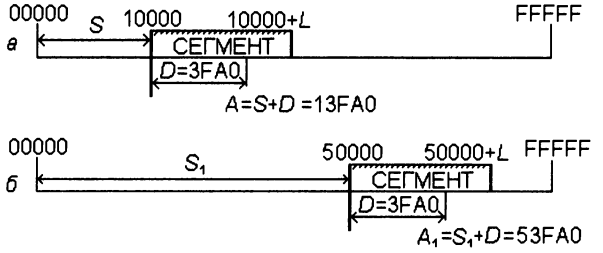


Рис. 4.8. Перемещение сегмента в оперативной памяти

Механизм сегментации памяти устроен следующим образом. Каждая программа может использовать для размещения своих кодов один или несколько сегментов

оперативной памяти. Операционная система, выбирая для программ подходящие участки оперативной памяти, указывает адреса начальных байтов этих участков в качестве адресов сегментов, закрепляемых за той или иной программой. Для адресации внутри этих сегментов в программах используются не физические, абсолютные адреса полей памяти, а их внутрисегментные смещения, то есть относительные адреса, отсчитанные относительно начал сегментов.

В качестве примера рассмотрим ситуацию, изображенную на рис. 4.8. Пусть операционная система нашла для некоторой программы подходящий участок оперативной памяти с начальным адресом 10000_{16} и объявила его сегментом, выделенным программе, например, для хранения данных. Пусть эта программа использует для хранения некоторого кода поле с внутрисегментным смещением $3FA0_{16}$. Тогда этот код фактически размещается по физическому адресу $10000_{16} + 3FA0_{16} = 13FA0_{16}$ (рис. 4.8, *а*). Допустим теперь, что сегментом объявлен другой участок с начальным адресом 50000_{16}. Тогда обсуждаемый код окажется расположенным по другому физическому адресу, $50000_{16} + 3FA0_{16} = 53FA0_{16}$ (рис. 4.8, *б*), но на программном коде и на ее выполнении это никак не скажется, потому что в программе все указания на этот код даются только с помощью одного и того же смещения $3FA0_{16}$.

ВНИМАНИЕ

В сегментной адресации физический адрес поля памяти *A* задается двумя компонентами: адресом сегмента *S* и внутрисегментным смещением *D*. Адрес, заданный в такой форме, принято называть полным указателем и записывать в виде пары *S:D*, например, 10000:3FA0.

Проведенные рассуждения без изменений переносятся на программу, использующую произвольное количество полей, положения которых задаются внутрисегментными смещениями. Фактическое положение начала сегмента в оперативной памяти никак не влияет на задание любых относительных адресов внутри сегмента. Следовательно, все указания полей через внутрисегментные смещения в программе остаются неизменными при любом изменении положения сегмента в оперативной памяти. Весь сегмент как единое целое, со всеми входящими в него полями перемещается на другой участок оперативной памяти, и при этом программный код совершенно не меняется. Если же для указания используемых полей использовать абсолютные, физические адреса, то любое изменение положения программы в оперативной памяти неизбежно повлечет за собой изменение программного кода, что совершенно неприемлемо.

Итак, описываемый способ задания адресов полей внутри программ обеспечивает им нужное свойство перемещаемости. При этом перемещение программы, использующей те или иные сегменты памяти, осуществляется простой заменой начальных адресов сегментов в процессе ее загрузки. Если же при выборе начальных адресов сегментов учитывать и их длины, то можно избежать также и наложения сегментов друг на друга. Следовательно, физические адреса полей, даже при совпадении их внутрисегментных адресов, всегда окажутся разными, и по этой причи-

не исчезнет и опасность взаимного уничтожения одновременно выполняющихся программ и данных. Таким образом, рассмотренный механизм сегментации памяти исключает выявленные противоречия использования оперативной памяти одновременно выполняющимися программами.

В различных архитектурах компьютеров сегменты памяти обладают разными свойствами. В компьютерах, использующих процессор i8086, сегмент оперативной памяти:

- всегда имеет длину 64 Кбайт;
- обязательно начинается с адреса, кратного шестнадцати: $S \bmod 16 = 0$; другими словами, сегмент должен быть выровнен по началу параграфа.

Заметим, что при длине сегмента L = 64 Кбайт максимально возможное внутрисегментное смещение равно $D = FFFF_{16}$. В самом деле, для такого максимального смещения (с учетом того, что нумерация начинается с нуля) длина сегмента равна $FFFF_{16} + 1_{16} = 10000_{16} = 1 \cdot 16^4 = 65536$ байта = 64 Кбайт. Итак, все внутрисегментные смещения для процессора i8086 находятся в диапазоне от 0000_{16} до $FFFF_{16}$, и их принято всегда задавать четырьмя шестнадцатеричными числами.

4.1.6. Программная модель процессора i8086

Так же, как и во время работы с оперативной памятью, программист представляет себе процессор в идеализированном виде, содержащем лишь те его свойства, которые могут быть использованы при разработке и выполнении программ. Все остальные технические детали игнорируются. Таким образом, вместе с программной моделью оперативной памяти программист использует и программную модель процессора (рис. 4.9).

Основная функция процессора — программно управляемая обработка данных — складывается из двух компонентов: собственно действия по обработке данных и управления последовательностью их выполнения. Для этого в составе процессора предусмотрено АЛУ, некоторые начальные сведения о логической структуре и принципах работы которого приведены в 3.2.12, а также обсуждаемое в дальнейшем более подробно **устройство управления** (УУ). Арифметико-логическое устройство обеспечивает выполнение всех действий над данными, которые входят в систему команд процессора. А устройство управления «понимает» программу пользователя и в соответствии с ней организует нужный порядок выполнения этих действий.

Для ускорения выполнения машинных команд в составе процессора предусмотрен отдельный, самый быстродействующий в компьютере, **регистровый** уровень памяти, образованный из некоторого количества **регистров**.

ВНИМАНИЕ

Регистр — это устройство в составе процессора, служащее для кратковременного, промежуточного хранения кодов в процессе их обработки.

Рис. 4.9. Программная модель процессора i8086

Регистры процессоров по своим функциям аналогичны регистру микрокалькулятора, состояние которого отображается панелью индикации. Этот регистр хранит, например, текущее, набираемое на клавиатуре калькулятора слагаемое, а затем, после выполнения сложения, содержимое регистра заменяется полученной суммой. При переходе к сложению другой пары чисел содержимое регистра вновь меняется. Такое временное хранение исходных чисел и результатов необходимо в процессе выполнения любых вычислений. Отметим, что регистры могут находиться не только в процессоре — они используются для временного хранения кодов в любых устройствах компьютера, кроме оперативной памяти.

В регистр процессора, как и в слово оперативной памяти, можно записать двоичный код, который будет в нем храниться до тех пор, пока в этот регистр не будет записан другой код или пока компьютер не будет выключен. Отметим, что запись какого-либо кода в регистр процессора принято называть **загрузкой**. Код, находящийся в регистре, может быть использован, прочитан произвольное количество раз.

Обычно в состав процессора входит от нескольких десятков до нескольких сотен регистров. Каждый регистр процессора характеризуется длиной, измеряемой в количестве байтов, из которых он состоит. Как правило, регистры процессоров имеют длину, совпадающую с длиной машинного слова или же кратную этой длине. Используются регистры длиной 2, 4, 8 и 16 байтов. Такие регистры могут хранить код числа, одного или нескольких символов, машинной команды или адрес оперативной памяти. Совокупность всех регистров процессора принято называть его **программными ресурсами.**

Итак, суммируем различия между двумя уровнями памяти компьютера — оперативной памятью и регистрами процессора:

- оперативная память — это *самостоятельное* устройство компьютера, а регистры являются составной частью других его устройств;
- оперативная память хранит большое количество данных, необходимых находящимся в ней программам в течение всего времени их выполнения, а отдель-

но взятый регистр содержит код данного, код команды, адрес поля и т. д., используемый только для одного или нескольких действий одной из программ;

- скорость передачи данных у регистров в десятки и сотни раз превышает скорость работы оперативной памяти;
- объем оперативной памяти составляет миллионы и миллиарды байт, а суммарная емкость регистров — до нескольких сотен или тысяч байт.

Регистры процессоров обычно группируются по выполняемым ими функциям. Например, отдельную группу образуют регистры, предназначенные для хранения начальных адресов сегментов оперативной памяти. Имеются также группы регистров, которые используются процессором только для своих внутренних потребностей. Такие регистры не могут явно использоваться программистом в разрабатываемых программах ни для каких целей.

Те регистры процессора, которые могут быть задействованы программистом при написании программ, имеют собственные названия и/или буквенно-цифровые обозначения, которые облегчают работу с ними и обеспечивают их однозначное указание. Кроме того, за каждым регистром закрепляется двоичный код, с помощью которого этот регистр указывается в машинных командах. Например, в процессоре i8086 имеется регистр counter hexadecimal (счетчик шестнадцатеричный), название которого обычно сокращается до буквенного обозначения `cx`. За этим регистром закреплен двоичный код 001_2.

Регистры процессора i8086

Теперь приступим к обсуждению доступных программисту регистров процессора i8086. Поскольку эти регистры или их аналоги постоянно используются при разработке программ на машинном уровне, рекомендуется обстоятельно познакомиться с их функциями и запомнить их обозначения.

В состав процессора i8086 входят 14 доступных программисту регистров, в том числе (рис. 4.9):

- группа из четырех регистров **общего назначения** (РОН): `ax`, `cx`, `dx`, `bx`;
- группа из четырех **указательных** и **индексных** регистров: `sp`, `bp` и `si`, `di`;
- группа из четырех **сегментных** регистров: `cs`, `ds`, `ss`, `es`;
- регистр-**указатель** (счетчик) **команд** `ip`;
- регистр **флажков** `flags`.

Все указанные регистры имеют длину два байта. Кроме того, каждый из четырех регистров общего назначения может рассматриваться либо как один 16-битный регистр, либо как пара автономно используемых 8-битных.

Все регистры общего назначения, а также указательные и индексные регистры могут использоваться в программах произвольным образом для хранения адресов памяти, кодов (номеров) регистров, машинных команд и данных любой природы. Сегментные регистры могут содержать только адреса сегментов. Регистр `ip` (от Instruction Pointer — указатель команд) всегда содержит адрес (точнее,

внутрисегментное смещение) поля, в котором находится текущая или же следующая команда выполняемой программы, а в регистре `flags` (флажки) собраны биты, играющие роль индикаторов, признаков результатов выполнения действий, о которых шла речь в 3.1 при обсуждении внутреннего состояния компьютера. Эти биты принято называть **флажками** — отсюда и название регистра.

Кроме общих возможностей, каждый из регистров общего назначения, указательных и индексных регистров имеет свою собственную специализацию. Использование регистров специальным, заранее оговоренным способом позволяет во многих случаях уменьшить длину кода команды, а значит, и сократить программу. Кроме того, при этом уменьшается время выполнения команды и программы в целом.

В приведенных ранее обозначениях регистров общего назначения буква `x` взята из слова heXadecimal (шестнадцатеричный), буква `h` — из слова Height (старший), `l` — из слова Low (младший). В обозначениях указательных регистров буква `p` соответствует слову Pointer (указатель). Обозначения индексных регистров содержат букву `i`, взятую из слова Index (индекс), а обозначения сегментных — букву `s`, выбранную из слова Segment (сегмент). В дальнейшем описании обозначений буквы, имеющие указанный смысл, не расшифровываются.

Специализацией регистра **аккумулятора** `ax` является его использование как своеобразного сумматора, в котором накапливаются результаты вычисления сумм и вообще результаты любых вычислений для данных, занимающих слово памяти. Кроме того, этот регистр используется как обязательный элемент в операциях обмена данными. Каждый из байтов регистра `ax` имеет собственное обозначение и может использоваться независимым образом как восьмибитный регистр. Старший байт обозначается `ah`, а младший — `al`.

Регистр **базы** `bx` (от base — база) применяется для организации специального способа определения адреса, который называется *адресацией с базированием*. Этот способ используется, в частности, для машинной реализации данных сложной структуры, например данных комбинированного типа (записей). Старший `bh` и младший `bl` байты регистра `bx` могут использоваться как самостоятельные 8-битные регистры.

Специализацией регистра **счетчика** `cx` (от counter — счетчик) является организация циклов в программах. Он используется для хранения текущего значения параметра цикла. Старший `ch` и младший `cl` байты регистра `cx` могут использоваться как самостоятельные 8-битные регистры.

Специальная роль регистра **данных** `dx` (от data — данные) состоит в хранении чисел, участвующих в умножении и делении. В некоторых операциях ввода/вывода его использование является обязательным. Старший `dh` и младший `dl` байты регистра `dx` могут использоваться как самостоятельные 8-битные регистры.

Основная функция регистров **указателя стека** `sp` (от stack — стек) и **указателя базы** `bp` — хранение внутрисегментных смещений. Кроме того, они используются при работе с данными сложной структуры, для работы со стеком и в некоторых других случаях.

Специальностью регистров *индекс источника* `si` (от source — источник) и *индекс приемника* `di` (от detector — приемник) является хранение индексов элементов массивов в так называемых цепочечных операциях, связанных с обработкой последовательностей байтов памяти. Типичный пример такой операции — копирование массива символов. При этом регистр `si` используется для хранения индекса источника (оригинала), а регистр `di` — индекса приемника (копии).

В программе, выполняющейся процессором i8086, можно использовать от одного до четырех сегментов оперативной памяти:

- сегмент кода — для хранения программного кода;
- сегмент данных — для хранения кодов используемых программой данных;
- сегмент стека — для организации стека, в котором могут храниться коды фактических параметров подпрограмм, а также коды любых других данных этой программы;
- дополнительный сегмент данных — для хранения кодов данных при нехватке возможностей основного сегмента данных.

За каждым из сегментов памяти закреплен соответствующий сегментный регистр: регистр сегмента кода `cs` (от code — код), регистр сегмента данных `ds`, регистр сегмента стека `ss` и регистр дополнительного сегмента `es` (от extended — дополнительный). Эти регистры играют важнейшую роль в организации сегментной адресации. Операционная система, выбирая участки оперативной памяти для размещения используемых программой сегментов, автоматически загружает их адреса в соответствующие сегментные регистры `cs` и `ss`. Содержимое регистров `ds` и `es` в случае их использования программист должен формировать самостоятельно. В связи с этим они не могут использоваться ни для каких иных целей, кроме основной — хранения адресов сегментов.

Формирование физического адреса

Как только что установлено, сегментные регистры должны хранить адреса сегментов. Однако регистры процессора i8086 16-битные, а адрес сегмента, являющийся физическим адресом оперативной памяти, — 20-битный. Для разрешения этого противоречия в процессоре i8086 принята следующая схема сегментной адресации. Известно, что сегмент должен быть выровнен на границу параграфа, а его адрес всегда заканчивается нулем. Следовательно, при загрузке адреса сегмента в сегментный регистр достаточно задать только первые четыре шестнадцатеричные цифры адреса, подразумевая, что его последняя пятая цифра — нуль. Пусть например, сегмент размещается по физическому адресу 78100_{16}, тогда в сегментный регистр следует загрузить код 7810_{16}. Загрузка в шестнадцатеричные регистры процессора второго компонента физического адреса — внутрисегментного смещения, которое задается четырьмя шестнадцатеричными цифрами, — не вызывает осложнений.

Итак, для хранения адресов сегментов используются сегментные регистры `cs`, `ds`, `ss` и `es`, а для хранения внутрисегментных смещений — указательные `sp`, `bp`, индексные `si`, `di` регистры, а также регистр указателя команд `ip`. Поэтому

физический адрес может быть задан парой регистров, в которых находятся его компоненты. В принципе, эта пара может формироваться произвольным образом, однако существуют стандартные комбинации регистров, образующих содержащую адрес пару, в частности:

- пара регистров `cs:ip`, содержимое которых рассматривается как составные части физического адреса указателя очередной команды программы;
- пара регистров `ds:si` служит для задания полного указателя в сегменте данных;
- пара `es:di` служит для задания полного указателя в дополнительном сегменте;
- пары `ss:sp` и `ss:bp` используются для формирования структуры стека и задания адресов в его сегменте.

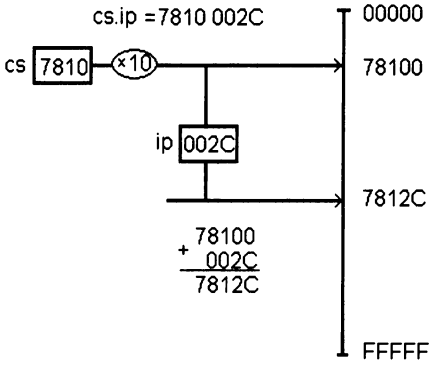


Рис. 4.10. Схема сегментной адресации процессора i8086

Рассмотрим изображенный на рис. 4.10 пример формирования физического адреса для процессора i8086. Пусть указатель задается в паре регистров `cs:ip`, при этом в регистре `cs` находится код 7810_{16}, а в регистре `ip` — код $002C_{16}$. Отметим, что такой указатель может быть записан и в виде обсуждавшейся ранее пары вида *S*:*D*, образованной из содержимого этих регистров, 7810:002C. Обратите внимание на то, что первый компонент указателя — адрес сегмента — в этом случае может задаваться только *первыми четырьмя цифрами*, находящимися в сегментном регистре.

Так как в сегментном регистре `cs` находятся только четыре первые цифры адреса сегмента 7810_{16}, то для получения *всего* адреса справа к ним следует приписать пятую цифру — 0. Эта операция может трактоваться как умножение кода из сегментного регистра на 10_{16}. Таким образом, в обсуждаемом примере адрес сегмента *S* равен $7810_{16} \cdot 10_{16} = 78100_{16}$. К этому адресу следует прибавить внутрисегментное смещение *D* из регистра `ip`. Получаем, что в паре регистров `cs:ip` с приведенным содержимым задан физический адрес $78100_{16} + 002C_{16} = 7812C_{16}$. Формальная запись этой операции имеет вид `A = [cs]` $\cdot 10_{16}$ `+ [ip]`, где заключенное в квадратные скобки название регистра представляет находящийся в этом регистре код, например, `[cs]` $= 7810_{16}$.

Следует обратить внимание на то, что принятая схема адресации успешно преодолевает противоречие между 16-битной разрядностью регистров процессора

и 20-битной разрядностью адресной шины. Кроме того, обеспечивается необходимое свойство перемещаемости программ, при этом для перемещения сегмента на другой участок оперативной памяти достаточно загрузить в его сегментный регистр другой адрес, и весь сегмент как единое целое, как твердое тело «переедет» в другое место памяти.

Регистр флажков flags

Регистр флажков содержит набор битов — флажков, которые играют роль индикаторов, признаков, характеризующих с некоторой точки зрения результаты выполняемых процессором действий. Процессор i8086 оперирует девятью различными флажками, шесть из которых, `of`, `zf`, `sf`, `af`, `pf` и `cf`, образуют группу флажков **состояния**, а остальные три, `df`, `if` и `tf`, входят в группу флажков **управления**. Флажки состояния характеризуют результаты выполнения действий и автоматически изменяются после выполнения процессором каждой команды, а флажки управления определяют режимы его работы.

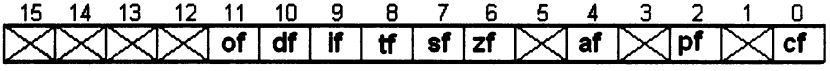


Рис. 4.11. Регистр флажков

На рис. 4.11 показано закрепление разрядов регистра за отдельными флажками, при этом неиспользуемые разряды регистра зачеркнуты, а в табл. 4.1 приведены условия, при выполнении которых отдельные флажки принимают значение 1. Если указанное в строке условие не выполнено, то соответствующий флажок получает нулевое значение.

Таблица 4.1. Флажки процессора i8086

Название	Разряд	Значение
`cf` (Carry Flag)	0	Флажок переноса из старшего бита результата. Равен 1, если был такой перенос. Является признаком выхода из диапазона представлений для беззнаковых чисел
`Pf` (Parity Flag)	2	Флажок контроля четности. Равен 1, если младший байт результата содержит четное количество единиц
`Af` (Auxiliary Flag)	4	Вспомогательный флажок переноса. Равен 1, если требуется коррекция сложения или вычитания BCD-данных
`zf` (Zero Flag)	6	Флажок нуля. Равен 1, если получен нулевой результат
`sf` (Sign Flag)	7	Флажок знака. Равен 1, если получен результат, который можно трактовать как отрицательное число
`of` (Overflow Flag)	11	Флажок переполнения. Равен 1, если был перенос из старшего или в старший бит результата и при этом состояние бита изменилось. Это признак выхода из диапазона представлений для знаковых чисел

продолжение

Таблица 4.1 (*продолжение*)

Название	Разряд	Значение
`tf` (Trace Flag)	8	Флажок режима трассировки. Равен 1, если процессор находится в режиме трассировки
`if` (Interrupt Flag)	9	Флажок прерывания. Равен 1, если аппаратные прерывания разрешены
`df` (Direction Flag)	10	Флажок направления. Равен 1, если цепочка обрабатывается в направлении возрастания адресов

Анализируя состояние флажков в программе, можно выбрать нужный режим ее выполнения, организовать контроль за операциями обмена с различными устройствами компьютера, проконтролировать допустимость полученных результатов, их принадлежность соответствующей области определения. Кроме того, такой анализ позволяет избежать появления в программе принципиально невыполнимых действий типа деления на 0, а также организовать в ней любые формы ветвлений и циклов. Далее смысл отдельных флажков рассматривается более подробно.

Отметим, что для флажков общепринятым является термин «установлен» как эквивалент выражения «принимает значение» или «имеет значение». Иногда оборот «флажок установлен» трактуется более узко — как закрепление за флажком значения 1. В этом случае для отражения его нулевого значения используется оборот «флажок снят».

Арифметические флажки zf и sf

Во многих задачах приходится сравнивать получаемые значения с нулем, а также определять их знак. Если в результате выполнения процессором текущей команды получен нулевой результат, то флажок `zf` (от Zero Flag — флажок нуля) получает единичное значение. В случае ненулевого результата флажок `zf` получает нулевое значение.

Если при выполнении команды получен результат, который может трактоваться как код отрицательного числа, то флажок `sf` (от Sign Flag — флажок знака) устанавливается в единицу; в противном случае он получает нулевое значение.

Флажки `zf` и `sf` являются основным инструментом для организации ветвлений и циклов в программах. Если, например, в программе нужно организовать деление на ветви по условию $x > 0$, то после вычисления значения x нужно проверить, удовлетворяют ли значения флажков `zf` и `sf` условию $\texttt{zf} = 0 \wedge \texttt{sf} = 0$.

Флажки контроля выхода из допустимого диапазона представлений cf и of

Флажок `cf` (от Carry Flag — флажок переноса) принимает значение 1, если во время выполнения процессором очередного действия осуществлен перенос из старшего бита результата. Это может быть 7, 15 или 31 бит у 1-, 2- и 4-байтного кода результата соответственно. В противном случае значение `cf` устанавливается равным нулю. Таким образом, если при выполнении некоторого действия,

например сложения двух кодов, рассматриваемых как беззнаковые числа в формате с фиксированной точкой, флажок `cf` оказался равным единице, это означает выход результата за пределы допустимого для рассматриваемого поля диапазона значений.

Флажок `of` (от Overflow Flag — флажок переполнения) также отслеживает переносы для старшего разряда результата, но немного по-другому. Если переноса в старший разряд не было и при этом не было переноса из него или же был перенос в старший разряд и при этом был перенос из него, то флажок принимает значение 0. Если был перенос в старший разряд, но переноса из него не было или же был перенос из старшего разряда, но переноса в него не было, то флажок устанавливается в значение 1. Различие между флажками `of` и `cf` в том, что значение `cf` зависит только от переноса из старшего разряда и не зависит от того, был или не был при этом перенос в этот разряд, в то время как для флажка `of` перенос в старший разряд имеет значение. Анализ флажка `of` позволяет отслеживать выход за пределы допустимого диапазона значений для знакового представления чисел в формате с фиксированной точкой.

Применение флажков `cf` и `of` для анализа допустимости результата проиллюстрировано в табл. 4.2. В ее втором столбце приведены примеры выполнения сложения для однобайтных двоичных кодов. Напоминаем, что один и тот же код может трактоваться и как беззнаковое, и как знаковое представление формата с фиксированной точкой (см. 2.5) и что для беззнаковых однобайтных кодов допустимым является диапазон чисел $0 \le x \le 255$, а для знаковых — диапазон $-128 \le x \le +127$ (см. 2.3.2). В третьем столбце таблицы показаны соответствующие действия в десятичном виде при трактовке кодов как беззнаковых, а в пятом — те же действия, но при трактовке этих же кодов как знаковых целых чисел. Четвертый и шестой столбцы содержат сформированные процессором при выполнении этих действий значения флажков `cf` и `of`.

Таблица 4.2. Установки флажков cf и of

№	Двоичный код	Беззнаковый	cf	Знаковый	of
1	$0000\ 0101_2$	5_{10}	0	+5	0
	$0010\ 0100_2$	36_{10}		+36	
	$0010\ 1001_2$	41_{10}		+41	
2	$0001\ 0001_2$	17_{10}	1	+17	0
	$1111\ 1001_2$	249_{10}		−7	
	$0000\ 1010_2$	10_{10}		+10	
3	$0000\ 1011_2$	11_{10}	0	+11	1
	$0111\ 1100_2$	124_{10}		+124	
	$1000\ 0111_2$	135_{10}		−121	
4	$1000\ 0111_2$	124_{10}	1	−121	1
	$1111\ 0101_2$	245_{10}		−11	
	$0111\ 1100_2$	124_{10}		+124	

В первой строке при выполнении сложения двоичных кодов нет переноса из старшего разряда — флажок `cf` равен 0. Соответствующее выполненному действию содержимое третьего столбца $5_{10} + 36_{10} = 41_{10}$ показывает, что беззнаковый результат для данного поля допустим. Так как при этом в старший разряд переноса также нет, то и флажок `of` устанавливается в нуль, показывая этим, что и знаковый результат допустим.

Во второй строке при выполнении сложения произведен перенос из старшего разряда. Следовательно, `cf` получает значение 1, которое сигнализирует о выходе за пределы допустимого диапазона при беззнаковой трактовке результата. Содержимое третьего столбца этой строки $17_{10} + 249_{10} = 101_{10}$ иллюстрирует этот выход. Но переносу из старшего разряда предшествовал перенос в этот разряд. Поэтому флажок `of` устанавливается в нуль, сигнализируя правильность действия при знаковой трактовке кодов, что подтверждается десятичным сложением $+17_{10} + (-7_{10}) = +10_{10}$.

В третьей строке перенос из старшего разряда отсутствует, следовательно, `cf` = 0 и действие в беззнаковой трактовке дает приемлемый результат: $11_{10} + 124_{10} = 135_{10}$. Но при этом был произведен перенос в старший разряд. Поэтому `of` = 1, что говорит о выходе за пределы допустимого диапазона в знаковой трактовке кодов: $+11_{10} + 124_{10} = -121_{10}$.

В последней, четвертой строке произведен перенос из старшего разряда, но перенос в него отсутствует. Значит, `cf` = 1 и одновременно `of` = 1. В любой трактовке результат выходит за пределы допустимого диапазона.

ВНИМАНИЕ

Процессор не различает коды чисел со знаком и без знака. Для него любой код представляет собой последовательность битов. Лишь программист придает кодам тот или иной смысл и с этих позиций организует нужную обработку кодов, используя для этого подходящие флажки.

Флажок вспомогательного переноса af

Кроме обсуждавшегося ранее формата с фиксированной точкой для кодирования целых чисел используется еще и так называемый упакованный двоично-десятичный, или BCD-формат. В этом формате код каждой цифры десятичного числа занимает четыре бита, или половину байта. При сложении или вычитании чисел в этом формате может возникнуть результат, который невозможно трактовать как число в формате BCD. В этом случае флажок `af` (от Auxiliary Flag — дополнительный флажок) устанавливается в 1. Анализ состояния этого флажка позволяет при необходимости выполнить необходимые корректирующие действия.

Флажок контроля четности pf

Процессор во время выполнения любых действий осуществляет контроль за общим количеством единиц в младшем байте результата. Если это количество четно, то флажок контроля четности `pf` (от Parity Flag — флажок паритета) получает значение 1. В противном случае он получает значение 0.

Флажки управления df и tf

В упомянутых ранее цепочечных операциях, когда действие выполняется над группой последовательных байтов памяти, они могут перебираться как в порядке возрастания адресов, так и в порядке их убывания. Этот порядок определяется установкой значения флажка направления `df` (от Direct Flag — флажок направления), относящегося к группе флажков управления. Программист имеет возможность с помощью специальной команды установить значение флажка в 0 (и тем самым определить перебор байтов цепочки в порядке убывания адресов) или же в 1 (и тем самым задать противоположный порядок).

В обычном, стандартном режиме работы процессор выполняет команды программы безостановочно друг за другом, пока не достигнет специальной команды, являющейся для него приказом на прекращение действий. Однако во время разработки программы на этапе ее **отладки**, когда осуществляется поиск допущенных ошибок, определяются их характер и местоположение, может быть использован специальный режим, который называется **трассировкой** программы. Особенность этого режима в том, что процессор останавливается после выполнения каждой очередной команды программы и на экран дисплея выдается полная информация о состоянии процессора и оперативной памяти (в том числе выводится содержимое всех регистров процессора). Шаг за шагом анализируя эту информацию, можно точно определить местоположение ошибки в программе, а затем и исправить ее. Переход в режим трассировки регулируется флажком `tf` (от Trace Flag — флажок трассировки). Если флажок установлен в 0, то программа выполняется в обычном безостановочном режиме. А его установка в 1 переводит процессор в режим трассировки.

Флажок запрещения прерывания if

С понятием прерывания мы познакомимся немного позднее (см. 4.2.7). А пока отметим, что механизм прерываний является важнейшей составной частью организации вычислений и обмена данными в компьютере. В некоторых ситуациях прерывания, запрашивающие обмен с некоторым внешним устройством или сигнализирующие о его завершении, следует временно запретить. Для этого нужно с помощью специальной команды закрепить за флажком `if` (от Interrupt Flag — флажок прерывания) значение 1. Запрещенные прерывания могут быть обработаны позднее, после того как флажок `if` получит нулевое значение.

4.2. Машинные команды процессора i8086

Прежде чем обсуждать систему команд процессора, следует более подробно выяснить, как устроена машинная команда, представляющая собой исчерпывающе точное указание процессору на выполнение отдельного действия, принадлежащего системе его команд.

4.2.1. Структура машинной команды

Вначале обратим внимание на то, каким образом задаются аналогичные действия, когда их исполнителем является человек. Типичными указаниями на выполнение действий, подобных командам компьютера, являются записи арифметических и алгебраических операций вида 4 + 3 = 7 или $c = ab$. Такое указание исполнителю-человеку содержит:

- обозначающий действие значок (+, –, ·, / и т. д.), который указывает, какое именно действие из всех доступных исполнителю (из тех, которые он умеет выполнять, из системы команд исполнителя) ему следует выполнить;
- числа, над которыми требуется выполнить действие. Они могут быть заданы непосредственно (3, 4) либо некоторым условным названием (a, b). Могут быть также заданы некоторые правила их определения;
- способ фиксации результата (7, c) выполненного действия.

Вполне очевидно, что указание для выполнения действия исполнителю — процессору компьютера — должно содержать не менее исчерпывающую информацию о действии. Естественным для компьютера способом его задания является двоичное кодирование всех элементов команды.

Каждому действию, принадлежащему системе команд процессора, ставится во взаимно однозначное соответствие двоичный код, который принято называть **кодом операции** (КОП). Данные (число, логическое значение, символ, строка бит, адрес поля памяти), участвующие в выполнении действия, обычно называются **операндами**. Операнд может быть задан непосредственно к команде, он может находиться в регистре процессора или в поле оперативной памяти. В любом случае операнд представлен некоторым двоичным кодом. Результат может быть помещен на хранение в один из регистров процессора или же в поле оперативной памяти. Если для операнда или результата используется регистр процессора, то необходимо указать, какой именно. В машинной команде это делается с помощью задания соответствующего регистру двоичного кода. Если используется поле памяти, необходимо указать его адрес, который также представляет собой двоичный код.

ВНИМАНИЕ

Машинная команда в целом представляет собой некоторый двоичный код, который так же, как и код операнда, может быть помещен в регистр процессора или в поле оперативной памяти. Для сокращения записи машинные команды обычно задаются в шестнадцатеричном виде.

Каждая машинная команда характеризуется *длиной*, которая равна количеству байтов, занятых ее кодом. Если машинная команда находится в поле оперативной памяти, то она характеризуется еще и *адресом*, которым считается адрес этого поля.

Для описания структуры кода команды, так же как и для описания кодов данных, используется понятие **формат команды**, являющееся разновидностью введенного в главе 2 общего понятия формата. Формат машинной команды представляет

собой исчерпывающе полный набор правил ее кодирования. Он определяет длину команды, распределение разрядов поля за отдельными ее элементами, количество и способы задания операндов, всевозможные признаки и параметры, уточняющие режим выполнения команды, и т. д.

Адресность машинных команд

Единственным обязательным элементом машинной команды является ее код. Кроме того, в составе команды обычно имеется несколько операндов, которые задаются либо непосредственно, либо указанием их местоположения (поле памяти или регистр процессора). Поскольку операнды в машинных командах чаще всего задаются определением их местоположения, соответствующий элемент кода команды принято называть **адресом**. В связи с этим при описании структуры команды мы будем обозначать ее операнды буквой А с каким-либо номером. Например, А1 — это обозначение первого операнда команды.

Как уже отмечалось, у разных машинных команд может быть различное количество операндов. Наибольшее количество операндов имеют *четырехадресные* команды со структурой КОП А1 А2 А3 А4. Первый (А1) и второй (А2) адреса задают операнды, например слагаемые. Третий адрес (А3) определяет место, куда следует записать результат, а четвертый операнд (А4) определяет адрес команды, которую процессор должен выполнить следующей.

Команды этого формата не очень распространены, так как в большинстве компьютеров принят естественный порядок выполнения команд программы, в соответствии с которым они выполняются последовательно друг за другом в том порядке, в каком находятся в оперативной памяти. В связи с этим адрес следующей команды задавать не нужно, так как он определяется простым сложением адреса и длины текущей команды. Этот принцип позволяет уменьшить количество операндов в команде и, следовательно, уменьшить ее длину.

- Наиболее естественную структуру имеют *трехадресные* команды — КОП А1 А2 А3. Такая команда содержит адреса операндов А1 и А2, а также адрес А3, по которому следует разместить результат ее выполнения. Систему трехадресных команд имели широко распространенная в свое время советская машина БЭСМ 4, ее аналоги М 220, М 222 и некоторые другие машины.
- С целью уменьшения длины команды из ее структуры можно исключить определяющий местоположение результата третий адрес. Результат при этом можно помещать по первому или по второму адресу команды, заменяя тот операнд, который в дальнейших вычислениях не требуется. Так получаются *двухадресные* команды со структурой вида КОП А1 А2. Двухадресной системой команд обладали, например, машины семейства «Минск».

Если пойти по пути дальнейшего упрощения структуры команды, то можно прийти к *одноадресным* командам вида КОП А1. Очевидно, что в команде с такой структурой может быть определен только один операнд. Положение другого операнда в одноадресных системах команд всегда фиксировано — он должен находиться в специальном регистре сумматора, который является составной частью

арифметико-логического устройства. Отсюда вытекает следующая схема задания такого, например, действия, как сложение. Вначале с помощью специальной команды одно из слагаемых заносится в регистр сумматора. Адрес другого слагаемого определяется в команде сложения. Результат остается в регистре сумматора, подготавливая тем самым выполнение следующей команды. Одноадресной была, например, система команд у лучшей в Европе в конце 1960-х гг. советской машины БЭСМ 6.

И наконец, следует упомянуть еще об одном возможном варианте — о *безадресных* командах, которые состоят только из кода операции. Операнды таким командам либо вообще не нужны, либо всегда строго фиксированы и не могут быть изменены ни при каких обстоятельствах. К безадресным относится, например, упоминавшаяся ранее команда остановки, прекращающая выполнение программы.

Способы адресации

Способы задания операндов в команде принято называть **адресацией**. Существует несколько десятков различных способов адресации, но, в принципе, все эти способы являются различными вариантами четырех основных:

- *непосредственная адресация* означает, что сам операнд (точнее, его код) включается в машинную команду как ее составная часть;
- *регистровая адресация* означает, что операнд находится в регистре процессора, а в команде указывается код этого регистра;
- *прямая адресация* означает, что операнд находится в поле оперативной памяти, а в команде указан адрес этого поля;
- *косвенная адресация* означает, что операнд также находится в поле оперативной памяти, а в команде содержатся некоторые элементы, по которым однозначно определяется адрес этого поля.

Отметим, что непосредственная адресация при всей ее простоте имеет ограниченные возможности применения, так как при любом изменении операнда приходится изменять и содержащую его код машинную команду. Это означает изменение и программы в целом, что, вообще говоря, весьма нежелательно, так как в большинстве случаев требует ее повторной подготовки к выполнению (например, повторной трансляции). Непосредственно в машинной команде могут быть заданы только такие операнды, которые не меняются при любых выполнениях программы.

Использование для задания операндов регистровой адресации является самым эффективным и самым простым способом. Однако, как мы знаем, процессор i8086 обладает малым количеством регистров. Поэтому неизбежны сложности при организации вычислений с большим количеством различных данных. Кроме того, в начале выполнения программы все ее данные размещаются в оперативной памяти. Следовательно, в любом случае необходимы способы адресации, обеспечивающие возможность выборки операндов из полей оперативной памяти. Такими возможностями обладают прямая и косвенная адресация. В прямой адресации адрес поля памяти прямо указывается в команде, а в косвенной его необходимо

определить по заданной в команде информации. Косвенная адресация по сравнению с прямой обеспечивает большую гибкость, необходимую при работе с данными сложной структуры, такими как массивы и записи,

В связи с введением косвенной адресации приходится различать адрес операнда из команды $A_{ком}$ и адрес, по которому происходит фактическое обращение в память, $A_{исп}$.

ВНИМАНИЕ

Адрес, заданный в команде, $A_{ком}$, представляет собой указанный в команде адрес поля или элементы, по которым он определяется. Исполнительный, или физический, адрес $A_{исп}$ представляет собой адрес, по которому производится фактическое обращение в оперативную память в момент выполнения команды.

При использовании косвенной адресации по заданному в команде адресу $A_{ком}$ производится вычисление исполнительного адреса: $A_{исп} = f(A_{ком})$. При этом функция f определяет вид косвенной адресации. При использовании прямой адресации эти адреса совпадают, поэтому прямую адресацию можно считать частным случаем косвенной, полагая f равной тождественной функции.

Система команд процессора i8086 смешанная, она содержит безадресные, одно- и двухадресные команды. При этом в двухадресных командах оба операнда не могут одновременно находиться в оперативной памяти. Один из них должен быть задан непосредственно либо выбираться из регистра процессора. Невозможны также команды, которые содержат оба операнда непосредственно в команде.

Машинный и ассемблерный форматы команд

В качестве конкретного, довольно простого примера машинной команды рассмотрим одноадресную команду с кодом $0100\ 0101_2$ или 45_{16}. Первые пять битов этой команды, 01000_2, являются кодом операции, а последние три, 101_2, — кодом регистра процессора bp. По этой команде осуществляются следующие действия:

- из регистра bp выбирается двухбайтный код и переносится в АЛУ;
- к выбранному коду прибавляется 1;
- результат записывается назад в регистр bp.

Таким образом, машинная команда 45_{16} увеличивает на единицу число, код которого находится в регистре bp. Это действие называется **инкрементом** операнда.

Запись команды в двоичном или шестнадцатеричном виде называется **машинным форматом** команды. Человеку записывать и воспринимать команды в таком виде очень сложно. Это требует значительных усилий по запоминанию машинных кодов, особенностей задания адресов, деталей выполнения действий и т. д. Поэтому разработан специальный способ записи машинных команд в символьном виде, который намного удобнее для человека. Вместо шестнадцатеричных или двоичных кодов в этом виде используются так называемые **мнемокоды**, то есть коды, удобные для запоминания. Мнемокоды представляют собой специально подобранные буквенно-цифровые обозначения, целые слова или сокращения

какого-либо естественного языка, например английского или русского. Так, мнемокод `inc` является сокращением от слова increment — приращение, увеличение значения на некоторую фиксированную величину. В данном случае имеется в виду увеличение на единицу. Мнемокод `inc` соответствует коду операции 01000_2, а мнемокод `bp` заменяет в команде код регистра 101_2. Следовательно, машинной команде 45_{16} соответствует символьная запись вида `inc bp`, по которой можно довольно легко понять осуществляемое командой действие — инкремент содержимого регистра `bp`. Догадаться об этом, глядя только на машинный код команды 45_{16}, по-видимому, невозможно. Запись команд с использованием мнемокодов называется **ассемблерным форматом**.

В табл. 4.3 сравниваются машинный и ассемблерный форматы одной и той же команды процессора. В последнем столбце приведена характерная для алгоритмических языков типа Паскаль запись выполняемого командой действия. Такую условную форму мы будем использовать в дальнейшем при кратком описании действий машинных команд.

Таблица 4.3. Сравнение машинного и ассемблерного форматов

Машинный формат		Ассемблерный формат	Действие
Двоичный	Шестнадцатеричный		
$0100\ 0101_2$	45_{16}	`inc bp`	`bp := bp + 1`

За каждой машинной командой закрепляется какой-либо мнемокод ассемблерного формата. Использование таких мнемокодов и специальных правил записи операндов значительно облегчает человеку написание и чтение программы. Но процессор понимает только язык двоичных кодов, поэтому возникает необходимость в переводе с языка мнемокодов, понятного человеку, на язык двоичных кодов, который понятен процессору. Так как между машинными командами и мнемокодами существует соответствие, а правила записи операндов довольно простые, такой перевод может быть автоматизирован. Для его выполнения разработаны специальные программы, которые в отечественной литературе ранее назывались **автокодами**. В настоящее время общепринятым для такого рода программ является название **ассемблер**[1]. Программа-ассемблер как бы собирает, монтирует машинную программу на основании заданной программистом последовательности мнемокодов. Этот процесс принято называть **ассемблированием**, **компиляцией** (от compile — собирать) или **трансляцией** (от translate — переводить).

Впоследствии ассемблером стали называть не только программу, которая осуществляет перевод на машинный язык, но и способ записи программ в такой форме — язык Ассемблер. Итак, термин «ассемблер» имеет два значения:

- язык программирования, то есть совокупность правил написания программ, в которых для записи действий используются мнемокоды машинных команд;
- программа, осуществляющая ассемблирование, то есть автоматический перевод на машинный язык программ, написанных на языке Ассемблер.

[1] Термин «ассемблер» произошел от англ. assemble — собирать, монтировать, соединять.

Поскольку процессоры различных машин обладают различными системами команд, то языки Ассемблер и сами программы-ассемблеры для различных компьютеров отличаются друг от друга.

Для каждой системы команд в сопроводительной технической документации приводятся коды всех входящих в нее команд, их названия и характерные особенности. Фрагмент системы команд процессора i8086 помещен в табл. 4.4. По нему видно, что одно и то же действие может быть представлено несколькими командами, с разными модификациями исполнения. Данная в таблице характеристика команды определяет ее адресность и модификацию (см. 4.2.3).

Таблица 4.4. Фрагмент системы команд процессора i8086

Название	Мнемокод	Код операции	Характеристика
Пересылка	`mov`	100010	Двухадресная
		1011	Непосредственная регистровая
		1100011; 000	Непосредственная общая
Сложение	`add`	000000	Двухадресная
Вычитание	`sub`	001010	Двухадресная
Инкремент	`inc`	01000	Одноадресная регистровая
		1111111; 000	Одноадресная общая
Декремент	`dec`	01001	Одноадресная регистровая
		1111111; 001	Одноадресная общая
Возврат в ОС	`retf`	11001011	Безадресная

4.2.2. Безадресные команды

Как известно, безадресные команды состоят только из кода операции (рис. 4.12, *а*). Они занимают один байт памяти. В табл. 4.5 приведены примеры безадресных машинных команд.

Таблица 4.5. Примеры безадресных команд

Машинный формат		Ассемблерный формат	Действие
$0110\ 0000_2$	60_{16}	`pusha`	Записать все регистры в стек
$0110\ 0001_2$	61_{16}	`popa`	Выбрать все из стека в регистры
$1111\ 0100_2$	$F4_{16}$	`hlt`	Остановить выполнение программы

Команда 60_{16} (мнемокод `pusha` (от push all — продвинуть все)) в строго определенном порядке записывает (продвигает) в стек содержимое всех регистров процессора, а команда 61_{16} (мнемокод `popa` (от pop all — появиться всем)) выполняет обратное действие — выбирает данные из вершины стека и загружает их в соответствующие регистры процессора. С командой $F4_{16}$ (мнемокод `hlt` (от halt —

останавливаться)) мы уже сталкивались — она прекращает выполнение программы. Очевидно, что всем этим командам операнды либо вообще не нужны, либо они строго фиксированы и никогда не изменяются.

4.2.3. Одноадресные команды

В системе команд процессора i8086 предусмотрены два варианта одноадресных команд: регистровая — более простая, но с ограниченными возможностями — и общая модификация.

Регистровая модификация

Регистровая модификация одноадресных команд может оперировать только с двухбайтными операндами, которые, к тому же, могут находиться только в регистрах процессора. Эти команды всегда занимают один байт памяти.

Структура одноадресных команд в регистровой модификации очень проста (рис. 4.12, *б*). Команда состоит из кода операции КОП, который занимает старшие пять бит байта, и трехбитного адреса операнда `reg`, который может быть только кодом регистра процессора. Анализируя рассмотренную ранее команду 45_{16} (`inc bp`), можно заметить, что она имеет именно этот формат. Возможные значения параметра `reg` команды и их смысл представлены в табл. 4.6.

Таблица 4.6. Кодировка параметра reg

000_2	001_2	010_2	011_2	100_2	101_2	110_2	111_2
ax	cx	dx	bx	sp	bp	si	di

Зная код операции и коды регистров, можно без труда формировать машинные команды этого формата. Например, мы уже выяснили, что команда инкремента имеет код операции 01000_2 и мнемокод `inc`. В системе команд процессора имеется также команда **декремента** с кодом операции 01001_2 и мнемокодом `dec`, которая уменьшает значение операнда на единицу (см. табл. 4.4). Пусть операнд находится в регистре `bx`, код `reg` которого равен 011_2. Тогда команда `inc bx`, выполняющая инкремент этого операнда, имеет код 01000 | 011_2 или 43_{16}, а команда `dec bx`, которая обеспечивает его декремент, имеет вид 01001 | 011_2 или $4B_{16}$. В табл. 4.7 приведен еще ряд примеров одноадресных команд. Для удобства их анализа в двоичном виде команд код операции отделен от кода регистра вертикальной чертой, которая, конечно же, в реальной машинной команде отсутствует.

Таблица 4.7. Примеры одноадресных команд в регистровой модификации

Машинный формат		Ассемблерный формат	Действие
01000 | 001_2	41_{16}	`inc cx`	`cx := cx + 1`
01001 | 001_2	49_{16}	`dec cx`	`cx := cx - 1`
01000 | 111_2	47_{16}	`inc di`	`di := di + 1`
01001 | 111_2	$4F_{16}$	`dec di`	`di := di -1`

Общая модификация

Операнды одноадресных команд в общей модификации могут быть длиной как один, так и два байта, они могут находиться и в регистрах процессора, и в оперативной памяти. В табл. 4.8 сравниваются возможности модификаций одноадресных команд.

Таблица 4.8. Сравнение модификаций одноадресных команд

Модификация	Длина операнда	Местоположение операнда
Регистровая	Только 2 байта	Только регистр процессора, `reg` — код регистра
Общая	1 или 2 байта	`mod`=112 — регистр процессора, `r/m` — код регистра
		`mod` ≠ 112 — поле памяти, `r/m` — способ адресации

Как видно из таблицы, возможности команд в общей модификации значительно шире, чем в регистровой. Но при этом и структура команды (рис. 4.12, *в*) значительно сложней. Сплошной линией на рисунке ограничены байты, присутствие которых в команде обязательно, а пунктиром — дополнительные, необязательные байты префикса и смещения, включаемые в команду по желанию программиста. Таким образом, одноадресные команды в общей модификации могут иметь длину от 2 до 5 байтов.

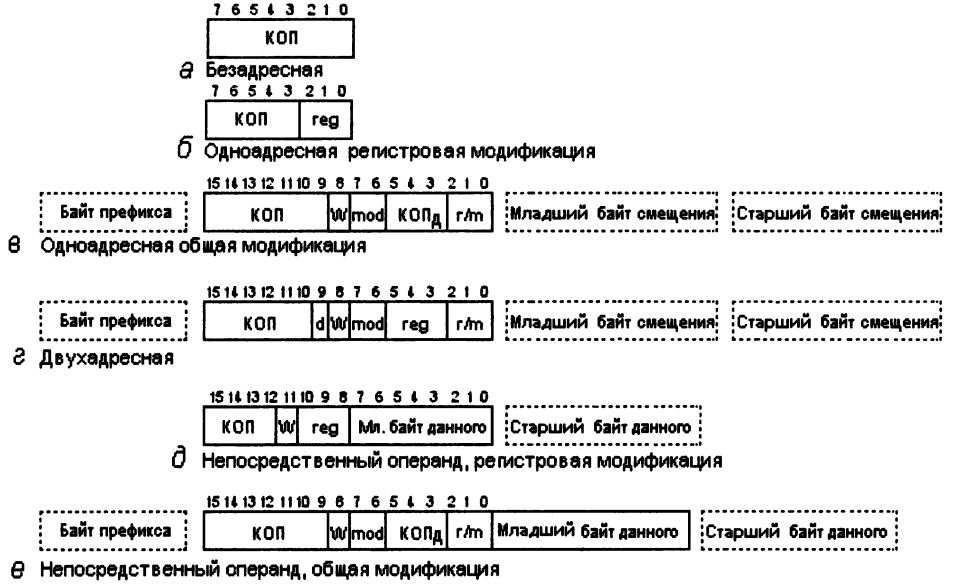


Рис. 4.12. Форматы машинных команд процессора i8086

Рассмотрим структуру обязательных байтов команды. Использованные на рис. 4.12 обозначения имеют следующий смысл:

- КОП (7 битов) — код операции (основная часть кода);
- $КОП_д$ (3 бита) — дополнительный код операции (уточняющая часть кода);

- W (1 бит) — признак длины операнда, 0 — операнд однобайтовый, 1 — операнд двухбайтовый;
- mod (от modification — модификатор, 2 бита) — параметр, определяющий местоположение операнда;
- r/m (от register/memory — регистр/память, 3 бита) — параметр, уточняющий адресацию операнда.

Код операции в командах этого формата разбит на две части, КОП и $КОП_д$, которые занимают соответственно разряды с 9-го по 15-й и с 0-го по 2-й двух основных байтов команды. Находящийся в 8-м разряде признак W определяет длину операнда команды. Если W = 0, то команда работает с однобайтовым кодом, а при W = 1 — с двухбайтовым. Занимающий 6-й и 7-й разряды параметр mod определяет местоположение операнда и уточняет структуру команды, включает она дополнительные байты смещения или нет. Если mod = 112, то операнд находится в регистре процессора, и тогда параметр r/m определяет его код. В противном случае, при $mod \neq 11_2$, операнд находится в оперативной памяти, а параметр r/m уточняет способ косвенной адресации.

Случай mod = 11_2

Вначале обсудим вариант, когда mod = 112 и параметр r/m определяет регистр процессора. В табл. 4.9 представлена его кодировка, которая зависит от заданной параметром W длины операнда. Заметим, что кодировка параметра reg (см. табл. 4.6) совпадает с кодировкой параметра r/m для W = 1 и mod = 112 (табл. 4.9). Длина команды может быть равной 2 или 3 байта, так как в этом случае два дополнительных байта смещения в команду не включаются.

Таблица 4.9. Кодировка параметра r/m при mod = 11_2

r/m	W = 0	W = 1	r/m	W = 0	W = 1
000	al	ax	100	ah	sp
001	cl	cx	101	ch	bp
010	dl	dx	110	dh	si
011	bl	bx	111	bh	di

Рассмотрим пример формирования команды при mod = 112. По табл. 4.4 находим, что команда инкремента inc в общей модификации имеет коды КОП = 1111111_2 и $КОП_д = 000_2$. Выберем значение параметра r/m = 1112. Тогда при W = 0 команда принимает вид КОП | W | mod | $КОП_д$ | r/m = 1111111 | 0 | 11 | 000 | 111_2 = 1111 1110 1100 0111_2 = FE $C7_{16}$ (в подготовительных записях элементы команды отделены друг от друга вертикальной чертой). По табл. 4.8 находим, что при W = 0 код r/m = 1112 определяет регистр bh, поэтому ассемблерский формат команды inc bh. По этой команде процессор выбирает из регистра bh однобайтовый код, выполняет его инкремент и результат записывает назад в регистр bh.

Если же W = 1, то код 111_2 определяет регистр di, и машинная команда принимает вид 1111111 | 1 | 11 | 000 | 111_2 = 1111 111 1100 0111_2 = FF $C7_{16}$ или inc di. В этом случае инкременту подвергается двухбайтовый код из регистра di. Последнее действие может быть задано и с помощью занимающей всего один байт регистровой модификации команды 01000 | 111_2 = 0100 0111_2 = 47_{16}. Заметим, что и для регистровой 47_{16}, и для общей модификации FF $C7_{16}$ этой команды ассемблерный формат inc di один и тот же.

Сравнение возможностей регистровой и общей модификаций показывает, что при написании программ можно использовать только команды в общей модификации. Однако выполняющие те же действия команды в регистровой модификации, во-первых, короче, во-вторых, имеют более простую структуру, а значит, выполняются быстрее, чем в общей модификации. Таким образом, использование регистровой модификации делает программу в целом более короткой и более эффективной.

Случай mod ≠ 11

Если параметр mod ≠ 11, то операнд находится в оперативной памяти. Физический (исполнительный) адрес A поля памяти вычисляется по уже известному соотношению

$$A = [S] \cdot 10_{16} + D_s, \tag{4.1}$$

где $[S]$ — содержимое сегментного регистра ss или ds; D_s — внутрисегментное смещение, которое задается как сумма не более чем трех компонентов:

$$D_s = [B] + [I] + D_k, \tag{4.2}$$

где $[B]$ — содержимое базового регистра bx или bp; $[I]$ — содержимое индексного регистра si или di; D_k — смещение, заданное в команде одним или двумя дополнительными байтами, которые показаны на рис. 4.12 пунктирной линией справа от основных байтов команды. Каждое из слагаемых в выражении для внутрисегментного смещения может отсутствовать, но не все сразу. Заданное таким образом внутрисегментное смещение D_s принято называть *эффективным адресом.*

Если внутрисегментное смещение зависит от регистра bp, то для определения физического адреса по умолчанию выбирается сегмент стека, адрес которого находится в регистре ss. В остальных случаях выбирается сегмент данных с адресом, находящимся в регистре ds.

ВНИМАНИЕ

Оборот «по умолчанию» означает автоматический выбор значения, параметра или действия, который реализуется, если ничего другого явно не указывается.

Набор слагаемых, входящих в выражение для эффективного адреса (4.2), определяется кодировкой параметров mod и r/m. При этом параметр mod регулирует наличие или отсутствие слагаемого D_k, а параметр r/m отвечает за комбинацию индексных и базовых регистров, содержимое которых включается в выражение (4.2).

В табл. 4.10 представлена используемая кодировка параметра mod. А в табл. 4.11 приведены кодировка параметра r/m при mod ≠ 11 и соответствующие этим кодам регистры процессора, которые участвуют в формировании адреса.

Таблица 4.10. Кодировка параметра mod

mod	Длина команды	Пояснение
002	2 или 3 байта	Операнд в поле памяти, дополнительное смещение в команду не включается и слагаемое D_k в выражении (4.2) отсутствует
012	3 или 4 байта	Операнд в поле памяти, смещение в команде D_k занимает один дополнительный байт, обозначенный на рис. 4.12 как «Младший байт смещения»
102	4 или 5 байтов	Операнд в поле памяти, смещение в команде D_k занимает два дополнительных байта, обозначенных на рис. 4.12 как «Младший байт смещения» и «Старший байт смещения»
112	2 байта	Операнд в регистре процессора

Таблица 4.11. Кодировка параметра r/m при mod ≠ 11_2

r/m	B	I	S
000_2	bx	si	ds
001_2	bx	di	ds
010_2	bp	si	ss
011_2	bp	di	ss
100_2	—	si	ds
101_2	—	di	ds
110_2	bp	—	ss
111_2	bx	—	ds

Для лучшего понимания обсуждаемого механизма адресации разберем конкретный пример. Как мы уже выяснили, в общей модификации операция инкремента inc имеет КОП = 1111111_2 и $КОП_д = 000_2$. Пусть первый дополнительный байт смещения содержит код 02_{16}, а следующий байт — код 01_{16}. Таким образом, в формируемой машинной команде все указанные элементы уже зафиксированы (рис. 4.13), и можно распоряжаться только параметрами W, mod и r/m, разряды которых на рисунке не заполнены.

Рис. 4.13. Пример формирования команды

Вначале рассмотрим возможные варианты команды при mod = 002, когда смещение из команды не учитывается, то есть байты, содержащие коды 02_{16} и 01_{16},

в команду не входят (кроме рассматриваемого далее особого случая mod = 002 и r/m = 1102 одновременно). Тогда команда имеет длину два байта (три при наличии байта префикса).

Некоторые варианты формирования команды приведены в табл. 4.12. Ее первый и второй столбцы содержат выбранные значения параметров r/m и W. После их выбора все разряды поля оказываются заполненными, а команда — полностью сформированной. Ее машинный, шестнадцатеричный формат приведен в третьем столбце. По выбранному значению параметра r/m в табл. 4.11 определяются участвующие в формировании физического адреса регистры процессора. Они приведены в четвертом столбце. Следующий столбец содержит рассчитанный по формулам (4.1) и (4.2) физический адрес, который используется для выборки операнда команды. И наконец, последний, шестой столбец содержит ассемблерный формат команды.

Для более подробного рассмотрения примеров из этой таблицы допустим, что регистры процессора содержат следующие коды: [bx] = 0010_{16}, [bp] = 0100_{16}, [si] = 0020_{16}, [di] = 0200_{16}, [ds] = 1000_{16}, [ss] = 2000_{16}, а байт памяти с адресом 10010_{16} — код 05_{16}.

В первой строке табл. 4.12 выбраны значения параметров r/m = 000_2 и W = 0. С учетом ранее определенных элементов команды КОП = 1111111_2, $КОП_д$ = 000_2 и mod = 00_2 получаем ее код 1111111 | 0 | 00 | 000 | 000_2 = 1111 1110 0000 0000_2 = = FE 00_{16}. По табл. 4.11 находим, что операнд находится в сегменте данных, адрес которого помещен в регистр ds, а внутрисегментное смещение определяется содержимым регистров bx и si. Сначала по формуле (4.2) находим D_s = [bx] + [si] = = 0010_{16} + 0020_{16} = 0030_{16}, а затем по формуле (4.1) получаем искомый адрес операнда A = [ds] · 10_{16} + D_s = 1000_{16} · 10_{16} + 0030_{16} = 10030_{16}. Итак, процессор при выполнении обсуждаемой команды FE 00_{16} и при указанных кодах в его регистрах обращается в оперативную память по адресу 10030_{16}, выбирает однобайтовый код, выполняет его инкремент и записывает результат назад в тот же самый байт памяти.

Таблица 4.12. Примеры формирования команд при mod = 00_2

r/m	W	16-ричный формат	Участвующие регистры	A	Ассемблерный формат
000_2	0	FE00	[ds], [bx], [si]	10030_{16}	inc byte ptr [bx+si]
000_2	1	FF00	[ds], [bx], [si]	10030_{16}	inc word ptr [bx+si]
001_2	0	FE01	[ds], [bx], [di]	10210_{16}	inc byte ptr [bx+di]
010_2	0	FE02	[ss], [bp], [si]	20120_{16}	inc byte ptr [bp+si]
011_2	0	FE03	[ss], [bp], [di]	20300_{16}	inc byte ptr [bp+di]
100_2	0	FE04	[ds], [si]	10020_{16}	inc byte ptr [si]
101_2	0	FE05	[ds], [di]	10200_{16}	inc byte ptr [di]
110_2^*	0	FE060201	[ds]	10102_{16}	inc byte ptr [0102]
111_2	0	FE07	[ds], [bx]	10010_{16}	inc byte ptr [bx]

Во второй строке выбрано то же самое значение параметра r/m = 000_2, но W = 1. Машинный код команды теперь имеет вид 1111111|1|00|000|000_2 = 1111 1111 0000 0000_2 = FF 00_{16}. Значения параметра W влияют только на длину выбираемого из поля памяти или регистра операнда, они никак не сказываются на формировании физического адреса. Поэтому операнд выбирается по тому же адресу 10030_{16}, но теперь он берется из двухбайтового поля.

Аналогичным образом определяется физический адрес и в следующих трех строчках таблицы. Различия проявляются только в том, что с помощью параметра r/m задаются другие регистры процессора и, следовательно, получаются другие значения адресов. Отметим только, что в формировании адреса для r/m = 010_2 и 011_2 участвует содержимое регистра bp. Поэтому операнд выбирается не из сегмента данных, а из сегмента стека, адрес которого находится в регистре ss.

Во всех рассмотренных случаях в формировании адреса участвует содержимое одного из базовых и одного из индексных регистров, поэтому такой способ косвенной адресации называется **адресацией с базированием и индексированием**.

Обсудим теперь строчки табл. 4.12, соответствующие значениям параметра r/m 100_2, 101_2 и 111_2. Для W = 0 им соответствуют машинные команды FE 04_{16}, FE 05_{16} и FE 07_{16}. В команде FE 04_{16} внутрисегментное смещение определяется только содержимым индексного регистра si: D_s = [si] = 0020_{16}. Такой вид косвенной адресации называется **адресацией с индексированием**. Поскольку операнд находится в сегменте данных, его физический адрес равен 10000_{16} + 0020_{16} = 10020_{16}. В команде FE 05_{16} также используется адресация с индексированием, но через регистр di: D_s = [di] = 0200_{16}. Следовательно, операнд находится по адресу 10200_{16}. А в команде FE 07_{16} смещение определяется только содержимым базового регистра bx: D_s = [bx] = 0010_{16} — это косвенная *адресация с базированием*, при этом операнд выбирается из поля с адресом 10010_2.

Отмеченная звездочкой строка табл. 4.12 соответствует значению r/m = 110_2, она относится к особому случаю кодирования параметров, к рассмотрению которого мы переходим. Внимательный анализ показывает, что все возможные комбинации значений параметров mod и r/m заняты различными видами косвенной адресации. Возможность закодировать с их помощью прямую адресацию отсутствует. Чтобы не вводить для прямой адресации отдельную модификацию (как это сделано для регистровой), разработчики процессора выбрали очень редко используемый на практике вариант косвенной адресации с базированием через регистр bp и соответствующую этому варианту комбинацию параметров mod = 00_2 и r/m = 110_2 закрепили за прямой адресацией. В этом случае внутрисегментное смещение D_s определяется *не содержимым регистров*, указанных в табл. 4.11, а только двумя байтами смещения из команды: D_s = D_k. Понятно, что описанный прием сделал соответствующий вариант косвенной адресации (базирование через регистр bp) недоступным[1].

С учетом сказанного мы можем разобраться в упомянутой строке табл. 4.12. В данном случае команда имеет длину 4 байта и код FE 06 02 01_{16}. Два дополнитель-

[1] Если все же возникнет необходимость организовать такую адресацию, можно задать mod = 01, r/m = 110 и равный 0016 дополнительный байт команды.

ных байта команды содержат смещение $D_k = 0102_{16}$. Здесь следует напомнить об обратном порядке записи кодов в полях оперативной памяти, принятом в процессорах Intel. Еще раз обратите внимание на рис. 4.12 и 4.13. В соответствии с этим принципом младшие биты смещения $0000\ 0010_2 = 02_{16}$ расположены в байте с младшим адресом, а старшие биты $0000\ 0001_2 = 01_{16}$ — в байте со старшим адресом. Следовательно, в «нормальном» порядке весь код имеет приведенный ранее вид 0102_{16}. Так как при использовании прямой адресации $D_s = D_k$, а адрес сегмента находится в ds, то операнд команды FE 06 02 01_{16} расположен по физическому адресу 10102_{16}.

В табл. 4.12 во всех (кроме второй) строчках таблицы длина операнда выбрана равной одному байту, и, соответственно, коды всех этих команд начинаются с байта FE_{16}. Точно так же, как во второй строке, во всех этих случаях можно взять длину операнда, равную двум байтам, и тогда коды команд будут начинаться с байта FF_{16}.

Рассмотрим теперь особенности ассемблерного способа записи этих команд. Вначале вспомним, что, например, по команде `inc bx` операнд выбирается из регистра bx, имя которого указано в команде. В результате выполнения этой команды код в регистре bx увеличится на единицу: $[bx] = 0011_{16}$. Здесь мы имеем дело с регистровой адресацией.

Если для задания операнда используется косвенная адресация, то в регистре находится не сам операнд, а его адрес или одно из слагаемых, входящих в выражение для определения адреса. В ассемблерном формате это отличие отображается с помощью квадратных скобок, в которые заключается название используемого регистра, например, `inc [bx]`. Такая запись означает, что используется косвенная адресация с базированием через регистр bx. В нашем примере $[bx] = 0010_{16}$ и адрес операнда равен 10010_{16} (последняя строка табл. 4.12). В результате по команде `inc [bx]` увеличивается на единицу *не код в регистре* bx, *а содержимое байта оперативной памяти* с адресом 10010_{16}. В табл. 4.13 сравниваются результаты выполнения команд `inc bx` и `inc [bx]` в предположении, что $[ds] = 1000_{16}$, а $[bx] = 0010_{16}$. В столбцах «До» и «После» показано содержимое адресов команд до и после выполнения команды, находящейся в первом столбце. Отчетливо видно, что в случае регистровой адресации действие выполняется над содержимым регистра, а в случае косвенной — над содержимым поля, адрес которого указан в регистре.

Таблица 4.13. Сравнение регистровой и косвенной адресации

Команда	[bx]		Байт памяти по адресу 10010_{16}	
	До	После	До	После
`inc bx`	0010_{16}	0011_{16}	05_{16}	05_{16}
`inc [bx]`	0010_{16}	0010_{16}	05_{16}	06_{16}

При использовании регистровой адресации длина операнда всегда определяется длиной используемого регистра. Так, в команде `inc bx` действие выполняется над двухбайтовым операндом, а в команде `inc bh` — над однобайтовым. Если же

операнд находится в оперативной памяти и используется прямая или косвенная адресация, то возникает необходимость в явном указании длины операнда. В машинных форматах команды содержат параметр `W`, который служит для такого определения. В ассемблерном формате для указания длины операнда служит параметр `ptr` (от pointer — указатель), перед которым словами `byte` или `word` задается длина 1 или 2 байта соответственно. Таким образом, обсуждаемая команда в ассемблерном формате принимает вид `inc byte ptr [bx]` для однобайтового операнда и `inc word ptr [bx]` — для двухбайтового.

В тех случаях, когда в косвенной адресации используются два регистра, например в адресации с базированием и индексированием, в ассемблерном формате команды следует указывать оба регистра: `inc byte ptr [bx][si]`. Запись можно давать и в указанном в табл. 4.12 упрощенном виде, когда оба регистра указаны в одной паре квадратных скобок: `inc byte ptr [bx + si]`.

В случае использования прямой адресации в квадратных скобках указывается не название регистра, а содержимое двух дополнительных байтов команды, определяющих смещение. Этот код принято задавать четырьмя шестнадцатеричными цифрами и (обратите внимание!) в «естественном» порядке следования байтов: для машинной команды FE 06 02 01_{16} ассемблерный формат имеет вид `inc byte ptr [0102]`. Для наглядности соответствующие фрагменты кодов в каждом из форматов подчеркнуты.

Пусть теперь `mod` = 01_2. Это означает, что команда содержит только один дополнительный байт, в котором находится младший байт смещения. В этом варианте команды имеют длину 3 или 4 байта. Примеры формирования команд для данного случая приведены в табл. 4.14.

Таблица 4.14. Примеры формирования команд при mod = 01_2

r/m	W	16-ричный формат	Участвующие регистры	A	Ассемблерный формат
000_2	1	$FE4002_{16}$	`[ds], [bx], [si]`	10032_{16}	`inc byte ptr [bx+si+02]`
000_2	2	$FF4002_{16}$	`[ds], [bx], [si]`	10032_{16}	`inc word ptr [bx+si+02]`
001_2	1	$FE4102_{16}$	`[ds], [bx], [di]`	10212_{16}	`inc byte ptr [bx+di+02]`
010_2	1	$FE4202_{16}$	`[ss], [bp], [si]`	20122_{16}	`inc byte ptr [bp+si+02]`
011_2	1	$FE4302_{16}$	`[ss], [bp], [di]`	20302_{16}	`inc byte ptr [bp+di+02]`
100_2	1	$FE4402_{16}$	`[ds], [si]`	10022_{16}	`inc byte ptr [si+02]`
101_2	1	$FE4502_{16}$	`[ds], [di]`	10202_{16}	`inc byte ptr [di+02]`
110_2	1	$FE4602_{16}$	`[ss], [bp]`	20102_{16}	`inc byte ptr [bp+02]`
111_2	1	$FE4702_{16}$	`[ds], [bx]`	10012_{16}	`inc byte ptr [bx+02]`

В целом рассуждения для этого варианта производятся в том же самом порядке, что и для предыдущего. Отличие состоит в том, что теперь во всех случаях в определении эффективного адреса участвует слагаемое D_k, выбираемое из одного дополнительного байта команды. Так, в первой строке табл. 4.13, в которой выбра-

ны значения параметров r/m = 000_2 и W = 0, с учетом ранее определенных элементов команды и mod = 01_2 получаем ее код в виде 1111111 | 0 | 01 | 000 | 000 | 00000010_2 = = 1111 1110 0100 0000 0000 0010_2 = FE 04 02_{16}. Внутрисегментное смещение определяется теперь не только содержимым регистров bx и si, но и частью кода команды: D_s = [bx] + [si] + D_k = 0010_{16} + 0020_{16} + 02_{16} = 0032_{16}. Такая адресация называется **адресацией с базированием, индексированием и смещением в команде**. Дальнейшее вычисление исполнительного адреса не отличается от рассмотренного ранее: A = [ds] · 10_{16} + D_s = 1000_{16} · 10_{16} + 0032_{16} = 10032_{16}. В ассемблерном формате этой команды дополнительное смещение из команды можно указывать двумя способами: так, как показано в табл. 4.13 — с помощью дополнительного слагаемого в квадратных скобках: `inc byte ptr [bx+si+02]`, или используя более короткий, но полностью эквивалентный способ, в котором смещение из команды указывается перед квадратными скобками: `inc byte ptr 02[bx+si]`.

Анализ остальных строк таблицы ничем не отличается от анализа, проведенного для первой строки. Отметим только, что значениям параметра r/m 100_2 и 101_2 соответствует **адресация с индексированием и смещением из команды**, а значениям 110_2 и 111_2 — **адресация с базированием и смещением из команды**.

Последний возможный вариант значения параметра mod = 10_2 отличается от предыдущего варианта mod = 01_2 только тем, что из команды берется не один, а два дополнительных байта смещения. Примеры формирования команды для этого случая приведены в табл. 4.15.

Таблица 4.15. Примеры формирования команд при mod = 10_2

r/m	W	16-ричный формат	Участвующие регистры	A	Ассемблерный формат
000_2	1	$FE800201_{16}$	[ds], [bx], [si]	10132_2	inc byte ptr [bx+si+0102]
000_2	2	$FF800201_{16}$	[ds], [bx], [si]	10132_2	inc word ptr [bx+si+0102]
001_2	1	$FE810201_{16}$	[ds], [bx], [di]	10312_2	inc byte ptr [bx+di+0102]
010_2	1	$FE820201_{16}$	[ss], [bp], [si]	20222_2	inc byte ptr [bp+si+0102]
011_2	1	$FE830201_{16}$	[ss], [bp], [di]	20402_2	inc byte ptr [bp+di+0102]
100_2	1	$FE840201_{16}$	[ds], [si]	10122_2	inc byte ptr [si+0102]
101_2	1	$FE850201_{16}$	[ds], [di]	10302_2	inc byte ptr [di+0102]
110_2	1	$FE860201_{16}$	[ss], [bp]	10202_2	inc byte ptr [bp+0102]
111_2	1	$FE870201_{16}$	[ds], [bx]	10132_2	inc byte ptr [bx+0102]

Как видно из приведенных примеров, принятая в процессоре i8086 схема адресации обладает очень высокой гибкостью, которая обеспечивает возможность реализации на уровне машинных кодов действий с данными любых сложных структур. Эта возможность основана на том, что замена содержимого участвующих в адресации базовых или индексных регистров позволяет без изменения команд программы обрабатывать коды из различных полей памяти, что как раз и необходимо при работе со структурированными данными.

Байт префикса замены сегмента

Во всех рассмотренных примерах операнд выбирался либо из сегмента данных, либо из сегмента стека. Этот выбор осуществлялся по уже упоминавшемуся действующему по умолчанию правилу: если в формировании эффективного адреса участвует содержимое базового регистра `bp`, то операнд выбирается из сегмента стека, во всех остальных случаях — из сегмента данных. Если такой выбор по каким-либо причинам программиста не устраивает, он может явно определить сегмент памяти, из которого следует выбрать операнд. Для этого в команду следует включить байт префикса замены сегмента. На рис. 4.14 он показан слева от основных байтов команды.

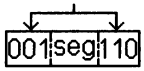


Рис. 4.14. Структура байта префикса замены сегмента

Байт префикса замены сегмента содержит специальную комбинацию битов, по которой его опознает процессор. Признаком префикса является наличие кодов 001_2 и 110_2 в пятом — седьмом и нулевом — втором разрядах байта соответственно, как показано на рис. 4.14. Третий и четвертый биты заняты кодом сегментного регистра `seg`, который должен участвовать в формировании адреса операнда вместо сегмента, определяемого по умолчанию. Кодировка параметра `seg` приведена в табл. 4.16.

Таблица 4.16. Кодировка байта префикса

Используемый сегмент	Параметр seg	Байт префикса целиком
`es`	00_2	$001 \mid 00 \mid 110_2 = 26_{16}$
`cs`	01_2	$001 \mid 01 \mid 110_2 = 2E_{16}$
`ss`	10_2	$001 \mid 10 \mid 110_2 = 36_{16}$
`ds`	11_2	$001 \mid 11 \mid 110_2 = 3E_{16}$

Например, в рассмотренной ранее команде FE 04 02_{16} операнд выбирается из сегмента данных. Если операнд нужно выбрать из дополнительного сегмента с адресом в `es`, то в команду нужно включить соответствующий байт префикса 26 FE 04 02_{16}. В ассемблерном формате такое переопределение задается названием нужного сегментного регистра, которое выставляется перед командой

и отделяется от нее двоеточием. В рассмотренном примере получим: `es: inc byte ptr [bx+si+02]`.

Включение байта префикса замены сегмента в команду увеличивает ее длину на один байт и требует дополнительной обработки, что, очевидно, снижает эффективность программы. Поэтому такую замену следует организовывать только в случае реальной необходимости.

4.2.4. Двухадресные команды

У двухадресных команд в системе команд процессора i8086 один из операндов *обязательно* должен находиться в регистре процессора, а второй может находиться как в регистре процессора, так и в поле оперативной памяти. Это ограничение отражается в структуре команды (см. рис. 4.12, *г*) следующим образом. Вместо дополнительной части кода операции $КОП_д$ команда содержит параметр `reg`, который, как в регистровой модификации одноадресных команд, определяет код регистра процессора, а второй адрес задается так же, как в их общей модификации — параметрами `mod` и `r/m`. Поскольку у двухадресных команд операнд может быть как однобайтовым, так и двухбайтовым, параметр `reg` кодируется в соответствии с табл. 4.9, то есть так же, как параметр `r/m` при `mod` = 11_2.

Представленный единственной частью код операции КОП занимает всего шесть битов. Зато в двухадресной модификации появился параметр `d`, влияющий на определение адреса записи результата. Если `d` = 0, то результат записывается по адресу, определяемому с помощью признаков `mod` и `r/m`, а при `d` = 1 он записывается в регистр процессора с кодом `reg`. Смысл и кодировка параметров `W`, `mod` и `r/m` по сравнению с одноадресными командами не изменились.

В качестве примера выясним действие машинной команды $00E8_{16}$. Вначале запишем команду в двоичном виде, а затем в соответствии с изображенной на рис. 4.12, *в* структурой двухадресной команды выделим составляющие ее элементы: $00E8_{16} = 0000001011101000_2 \rightarrow 000000 \mid 0 \mid 0 \mid 11 \mid 101 \mid 000_2$. Получаем, что КОП = 000000_2, `d` = 0, `W` = 0, `mod` = 11_2, `reg` = 101_2, `r/m` = 000_2. По табл. 4.4 находим, что кодом операции КОП = 000000_2 обладает команда сложения, имеющая мнемокод `add`. По табл. 4.9 выясняем, что при `W` = 0 параметр `reg` = 101_2 определяет восьмибитный регистр `ch`, а параметр `r/m` = 000_2 — регистр `al`. Таким образом, команда $00E8_{16}$ задает операцию сложения для восьмибитовых слагаемых, коды которых находятся в регистрах `ch` и `al`. Поскольку `d` = 0, результат направляется по адресу, заданному параметрами `mod` и `r/m`, то есть в регистр `al`. Условно действие этой команды можно записать в виде `al := al + ch`.

В связи с отсутствием в ассемблерном формате параметра `d`, определяющего адрес записи результата, действует правило, в соответствии с которым первым в команде указывается операнд, принимающий результат: `add ch, al`. Обратите внимание на то, что команда `add ch, al` выполняет действие `ch := ch + al`, которое задается машиной командой $02E8_{16}$, отличающейся тем, что в ней параметр `d` = 1. Сравнение этих вариантов команды проиллюстрировано в табл. 4.17

в предположении, что регистр al до выполнения команды содержит код 05_{16}, а регистр ch — код 07_{16}.

Таблица 4.17. Сравнение форматов команд

Машинный формат	Ассемблерный формат	Действие	al		ch	
			До	После	До	После
$00E8_{16}$	add al, ch	al := al + ch	05_{16}	$0C_{16}$	07_{16}	07_{16}
$02E8_{16}$	add ch, al	ch := ch + al	05_{16}	05_{16}	07_{16}	$0C_{16}$

Попутно заметим, что команды $01E8_{16}$ или add ax, bp и $03E8_{16}$ или add bp, ax оперируют с двухбайтовыми слагаемыми из регистров ax и bp.

4.2.5. Команды с непосредственным операндом

Как известно, в командах с непосредственной адресацией код операнда является частью кода команды. В системе команд процессора i8086 имеется две формы команд с непосредственным операндом. В регистровой форме, структура которой приведена на рис. 4.12, *д*, непосредственный операнд, находящийся в одном или двух дополнительных байтах команды, участвует в действии вместе с регистром процессора, который задан параметром reg. Байт префикса в такую команду никогда не включается. Поэтому команда имеет длину 2 или 3 байта в зависимости от определяемой значением параметра W длины операнда.

В качестве примера рассмотрим последовательность из трех байтов, содержащих код B7 0F $F0_{16}$ = 1011 0111 0000 1111 1111 0000_2. Выделение по схеме (см. рис. 4.12, *д*) отдельных элементов первого байта дает 1011 | 0 | 111_2. Получаем, что КОП = 1011_2, W = 0 и reg = 111_2. Следовательно, длина непосредственного операнда 1 байт, а длина всей команды — 2 байта. Таким образом, код команды представлен первыми двумя байтами B7 $0F_{16}$, а третий байт последовательности $F0_{16}$ в команду не входит. Из табл. 4.4 видно, что код операции КОП = 1011_2 имеет команда пересылки с мнемокодом mov (от move — движение). При W = 0 параметр reg = 111_2 определяет регистр bh, а код непосредственного операнда содержится во втором байте команды $0F_{16}$. По этой команде код $0F_{16}$ копируется из команды в регистр bh.

Отметим, что в ассемблерном формате команды код непосредственного операнда может быть задан в различных системах счисления. При этом запись операнда имеет вид mov bh, 0Fh — для шестнадцатеричной, mov bh, 15 — для десятичной и mov bh, 00001111b — для двоичной системы счисления. Выставляемые в конце операнда буквы h и b являются признаками используемой для задания операнда системы счисления. Десятичный операнд можно задавать с буквой d на конце или, как показано в примере, вообще без признака.

Пусть теперь исходная последовательность байтов содержит код BF 0F $F0_{16}$. Аналогичный проведенному ранее анализ показывает, что в этом случае W = 1. Следовательно, команда имеет длину три байта и все байты последовательности

представляют ее код, при этом непосредственный операнд занимает два последних байта команды 0F $F0_{16}$. Параметр reg = 111_2 при W = 1 определяет регистр di. Вследствие обратного порядка записи кодов данных в оперативной памяти младший дополнительный байт команды (с меньшим адресом) пересылается в младший байт регистра di, а старший дополнительный байт — в старший байт регистра. Таким образом, после выполнения команды регистр di содержит код $F00F_{16}$.

Ассемблерный формат команды mov di, 0F00Fh имеет особенность в записи шестнадцатеричного кода непосредственного операнда. Так как код $F00F_{16}$ начинается с шестнадцатеричной цифры F, транслятор может его спутать с каким-либо используемым в программе именем. Чтобы этого не произошло, перед кодами данных, которые начинаются с шестнадцатеричных цифр A, B, C, D, E и F, записывается нуль: 0F00Fh.

В общей форме непосредственный операнд участвует в действии вместе с полем оперативной памяти. Структура такой команды показана на рис. 4.12, *е*. Команда может занимать от двух до пяти байтов в зависимости от наличия или отсутствия байта префикса и от длины непосредственного операнда.

Отметим, что для этой модификации параметр mod может принимать только значение 00_2. Остальные значения параметра mod недопустимы, потому что при mod = 01_2 или 10_2 в дополнительных байтах команды должно размещаться смещение, а в обсуждаемой форме эти байты заняты непосредственным операндом.

В качестве примера рассмотрим код, занимающий последовательность из четырех байтов C6 07 0F $F0_{16}$. Его анализ показывает, что W = 0, и, следовательно, команда занимает только три байта последовательности: C6 07 $0F_{16}$. Код непосредственного операнда $0F_{16}$ занимает последний байт команды. Выделение остальных элементов команды дает следующий результат: КОП = 1100011_2, mod = 00_2, $КОП_д$ = 000_2, r/m = 111_2. Полученные основной и дополнительный коды операции принадлежат команде пересылки mov непосредственного операнда в поле памяти (табл. 4.4). Значения параметров mod = 00_2 и r/m = 111_2 соответствуют адресации с базированием через регистр bx: mov [bx], 0Fh. По этой команде код $0F_{16}$ записывается в поле памяти, адрес которого определяется содержимым регистров bx и ds.

Если рассматриваемая последовательность байтов содержит код C7 07 0F $F0_{16}$, то параметр W = 1 показывает, что вся последовательность содержит четырехбайтовый машинный код команды с ассемблерным форматом вида mov word ptr [bx], 0F00Fh, а ее последние два байта — код непосредственного операнда. По этой команде код $F00F_{16}$ записывается в оперативную память по тому же адресу, что и в предыдущем примере.

4.2.6. Схема работы процессора при выполнении машинной команды

Рассмотрим упрощенную схему выполнения процессором машинных команд программы. Эта схема на логическом уровне адекватна реальной работе процессора. Отличие состоит в том, что в реальном процессоре с помощью различных

технических приемов организуется параллельное, то есть одновременное выполнение нескольких этапов одной и той же команды или нескольких последовательных команд программы, за счет чего добиваются значительного ускорения работы процессора (см. 11.2).

Перед запуском программы на выполнение операционная система автоматически записывает в регистр cs адрес сегмента кода, который, как указывалось ранее, содержит код выполняющейся программы. Допустим, что операционная система занесла в регистр cs код 1000_{16}. Первые 256_{10} байтов сегмента кода операционная система всегда занимает вспомогательной информацией. Этот участок памяти принято называть **префиксом программного сегмента**, часто встречается также его обозначение PSP (от program segment prefix). Поэтому первый байт программы всегда имеет внутрисегментное смещение $0100_{16} = 256_{10}$. Как известно, для задания адресов в сегменте кода используется пара регистров cs:ip. Следовательно, операционная система перед началом выполнения программы должна записать в регистр ip код 0100_{16}. На рис. 4.15 изображена схема задания адреса первого байта программы при сделанном относительно содержимого регистра cs предположении.

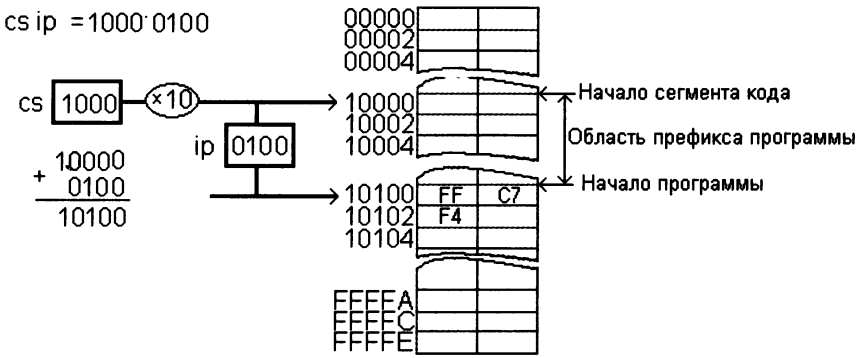


Рис. 4.15. Адресация начала машинной программы

На рис. 4.16, *а* слева изображена программная модель оперативной памяти, а справа — программная модель процессора в состоянии, которое мы будем считать исходным. Пусть в оперативной памяти в поле с адресом 10100_{16} находятся коды FF C7 $F4_{16}$, которые мы будем рассматривать как машинную программу, а регистры cs и ip содержат выбранные ранее коды 1000_{16} и 0100_{16} соответственно. Из всех остальных регистров процессора в данном примере нас интересует только содержимое регистра di, которое мы примем равным, например, $003A_{16}$. Работу процессора при выполнении машинной команды удобно разбить на несколько этапов.

Первый этап выполнения команды изображен на рис. 4.16, *б*. Указатель из пары регистров cs:ip определяет адрес, который всегда рассматривается процессором как адрес первого байта текущей команды выполняемой программы. В момент

запуска программы этот указатель, очевидно, представляет адрес ее первой команды. Он выбирается из пары cs:ip, выставляется на адресную шину и передается в оперативную память.

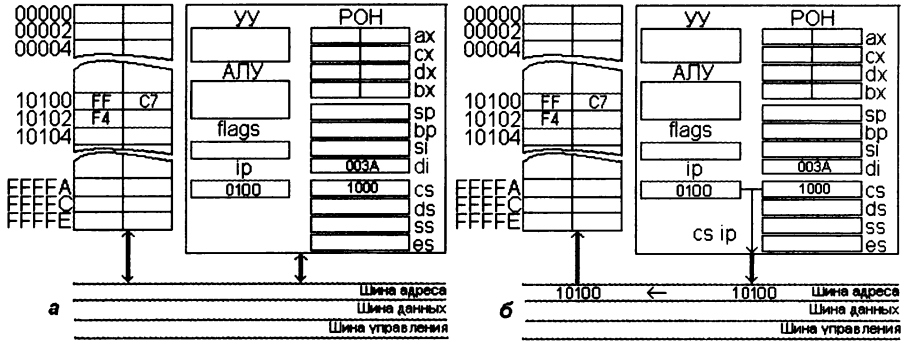


Рис. 4.16. Выполнение машинной команды: *а* — исходное состояние; *б* — первый этап

На втором этапе работы процессора содержимое поля, начинающегося с полученного оперативной памятью адреса, передается по шине данных в устройство управления процессора. Поскольку это содержимое рассматривается процессором как код машинной команды, возникает вопрос: сколько байтов поля должно быть передано в устройство управления? Мы знаем, что в системе команд процессора i8086 машинная команда может иметь длину от 1 до 5 байтов. Поэтому в устройство управления передается последовательность не менее чем из 5 байтов. Код из первого байта сравнивается с возможными значениями байта префикса замены сегмента (26_{16}, $2E_{16}$, 36_{16} и $3E_{16}$). Если этот код не является префиксом, он считается байтом команды, в котором находится код операции и по которому определяется ее общая длина. В противном случае информация о длине команды выбирается из следующего байта. Затем из переданной последовательности для анализа отбирается только необходимое количество байтов, соответствующее длине команды.

Следует обратить внимание на то, что байты кода машинной команды передаются в устройство управления в том же порядке, в котором они находятся в оперативной памяти. *Обратный порядок записи на коды машинных команд не распространяется.* Сфера действия обратного порядка записи — это коды данных, над которыми требуется выполнять арифметические операции, коды чисел в форматах с фиксированной или плавающей точкой, коды, используемые для формирования адресов, и т. д.

На третьем этапе устройство управления выполняет описанный ранее анализ, определяет длину команды, выделяет из переданной последовательности байтов код команды, а затем увеличивает содержимое регистра ip на ее длину. В рассматриваемом примере содержимое ip увеличивается на 2 и становится равным 0102_{16}, после чего в регистре ip оказывается адрес (точнее, внутрисегментное смещение) для следующей команды программы.

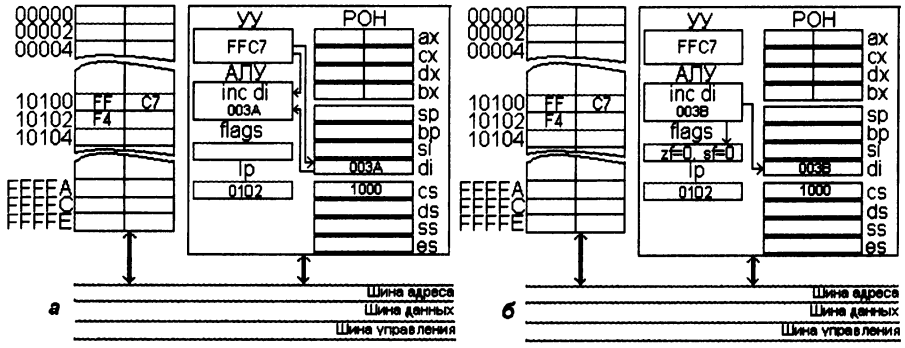


Рис. 4.17. Выполнение машинной команды: *а* — четвертый этап; *б* — пятый и шестой этапы

Четвертый этап выполнения команды показан на рис. 4.17, *а*. Устройство управления дешифрует команду, выделяя из ее кода все элементы, определяющие выполняемое действие, длину и адреса операндов. Затем в арифметико-логическое устройство передается информация об операции, которую необходимо выполнить, и по заданным в команде адресам запрашиваются операнды, которые также передаются в арифметико-логическое устройство. В рассматриваемом примере код FF $C7_{16}$ (`inc di`) задает выполнение инкремента содержимого регистра `di`. Устройство управления настраивает арифметико-логическое устройство на выполнение указанного действия и организует пересылку в него операнда $003A_{16}$ из регистра `di`.

В рассмотренном примере операнд команды находится в регистре процессора, что значительно упрощает ее выполнение. Если операнд находится в поле оперативной памяти, то устройство управления сначала передает на шину адрес операнда, затем оперативная память по шине данных возвращает в арифметико-логическое устройство код запрошенного операнда. Очевидно, что эти дополнительные действия значительно снижают скорость выполнения команды по сравнению с вариантом, когда операнд находится в регистре процессора. Поэтому следует *всегда, когда это возможно, использовать регистровую или непосредственную адресацию* и как можно реже обращаться за операндами в оперативную память.

Результаты выполнения пятого и шестого этапов изображены на рис. 4.17, *б*. На пятом этапе арифметико-логическое устройство выполняет заданное в команде действие. Его результатом является код $003B_{16}$. И на последнем, шестом этапе организуется определенная в команде пересылка результата и формируется соответствующее ему содержимое регистра флажков. В данном примере код $003B_{16}$ пересылается в регистр `di`. Так как получен ненулевой код, то флажок `zf` получает нулевое значение, `zf` = 0. Кроме того, код $003B_{16}$ не может трактоваться как код отрицательного числа, поэтому и флажок `sf` получает нулевое значение, `sf` = 0.

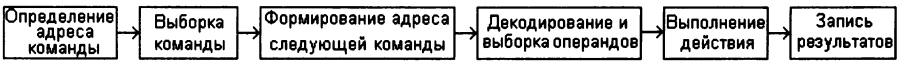


Рис. 4.18. Этапы выполнения машинной команды

На рис. 4.18 представлена рассмотренная последовательность этапов выполнения машинной команды процессором. Эта последовательность этапов работы процессора в точности повторяется в ходе выполнения следующей машинной команды. По адресу, определяемому указателем из пары регистров cs:ip, из оперативной памяти в устройство управления выбирается машинная команда. Команда дешифруется, определятся ее длина и соответствующим образом увеличивается содержимое регистра ip. После этого команда выполняется и результаты ее выполнения заносятся в заданное место хранения. Этот циклический процесс выполнения машинных команд обрывается при поступлении в процессор команды остановки с кодом $F4_{16}$. В данном примере она попадает в процессор уже на втором шаге.

4.2.7. Отладчик машинных программ debug

Обычно программист, работающий с языками высокого уровня типа Паскаль, С или Ява, не использует коды машинных команд, регистры и флажки процессора и т. д. Такой программист работает на совершенно другом уровне абстракции, взаимодействуя с аппаратурой компьютера, с его процессором и оперативной памятью через промежуточные уровни операционной системы и инструментальных средств используемой системы программирования. Даже программист, работающий с языком Ассемблер, имеет дело в основном с мнемокодами.

Одна из немногих возможностей напрямую работать с «живыми» машинными кодами команд, с полями оперативной памяти, регистрами процессора и т. д. состоит в использовании входящего в операционную систему MS DOS отладчика debug. Эта программа доступна и из операционных систем семейств Windows 9x/XP. Вызвать ее на выполнение можно следующим образом: командой Пуск ▸ Выполнить открыть окно Запуск программы, затем в поле ввода Открыть задать имя debug. Запускается отладчик в стандартном окне Windows (рис. 4.19[1]), работа с которым, в принципе, может быть заменена режимом отображения во весь экран. Но в некоторых версиях Windows этот режим неустойчив, поэтому рекомендуется оставаться в стандартном окне.

Программа debug предназначена для отладки программ на уровне машинных кодов. Полный набор возможностей этой программы можно найти в любом справочном руководстве по командам операционной системы MS DOS. В частности, отладчик debug обеспечивает:

- прямой ввод кодов программ и данных в выбранные поля оперативной памяти, причем коды могут вводиться как в машинном (шестнадцатеричном), так и в ассемблерном формате;
- редактирование содержимого полей памяти в шестнадцатеричном или символьном виде;
- запуск программ на выполнение в обычном режиме и в режиме трассировки;

[1] Содержимое рабочей зоны окна представлено на рисунке в инвертированном цвете.

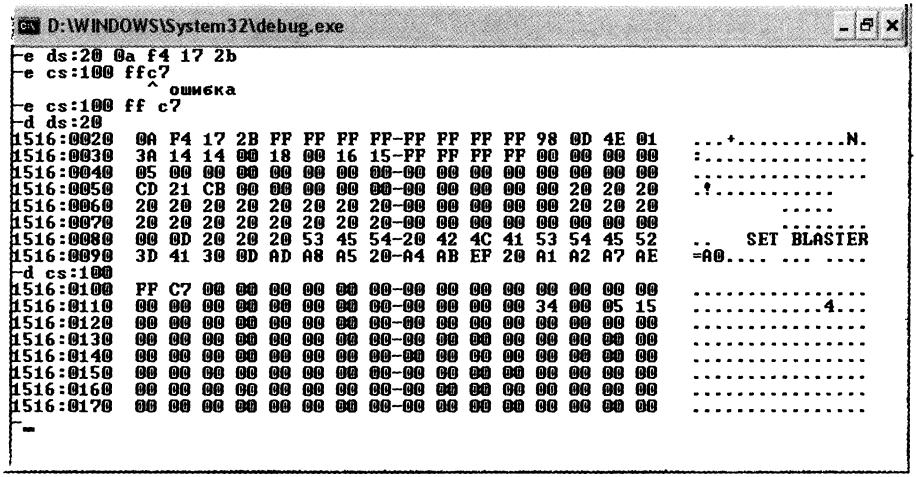


Рис. 4.19. Пример окна отладчика

- просмотр содержимого выбранного участка оперативной памяти;
- просмотр содержимого регистров процессора;
- целый ряд других режимов, которые обслуживают все связанные с отладкой программ действия.

Сразу после запуска отладчика он, так же как и операционная система MS DOS, переходит в режим ожидания команд пользователя, о чем говорит приглашение к вводу, которое имеет вид мигающей черточки в первой позиции текущей строки. Управление работой отладчика осуществляется с помощью его внутренних команд, каждая из которых вводится в соответствии с правилами работы с командной строкой операционной системы MS DOS. В частности, при вводе команд можно использовать клавишу Backspace для удаления последнего введенного символа. Набор команды завершается нажатием на клавишу Enter.

Из всей совокупности команд отладчика рассмотрим только пять команд, которые необходимы при дальнейшем обсуждении работы с отладчиком. Ввод кодов программ и данных в выбранный участок оперативной памяти, а также их редактирование в шестнадцатеричном виде можно выполнить с помощью команды E или e (от enter, можно использовать как строчные, так и прописные буквы), которая имеет формат

```
e <адрес поля> <данные>
```

Элемент команды `<адрес поля>` задается в виде пары *S:смещение*, где *S* — имя сегментного регистра, а $0 \le$ *смещение* $\le \mathrm{FFFF}_{16}$ — заданное шестнадцатеричным числом внутрисегментное смещение, причем ведущие нули в смещении можно опускать, например, `ds:20` или `cs:100`.

Элемент команды `<данные>` задается в виде последовательности шестнадцатеричных кодов, определяющих содержимое последовательности байтов памяти. Каждый очередной код, состоящий ровно из двух цифр, отделяется от предыду-

щего кода обязательным пробелом. При вводе кодов можно использовать как заглавные, так и строчные буквы. Например, по команде

```
e ds:20 0a f4 17 2b
```

последовательные байты сегмента данных со смещениями 0020_{16}, 0021_{16}, 0022_{16} и 0023_{16} заполняются указанными в команде шестнадцатеричными кодами `0a f4 17 2b`. Эта команда показана в первой строке окна отладчика на рис. 4.19.

Во второй строке окна приведен пример реакции отладчика на допущенную при вводе ошибку: программа отмечает знаком ^ первый, по ее мнению, неправильный символ команды и рядом выводит слово «ошибка». В данном случае ошибочно не введен пробел между кодами двух соседних байтов `ff` и `c7`. Ниже показана эта же команда в правильном варианте ввода.

Для изменения содержимого какого-либо байта или поля памяти с помощью команды `e` достаточно указать в ней адрес этого байта или поля и его новое содержимое.

Во время отладки довольно часто приходится просматривать содержимое некоторого участка оперативной памяти.

ВНИМАНИЕ

Отображение на экране дисплея или на бумаге содержимого некоторого участка оперативной памяти в виде шестнадцатеричных кодов, а также в символьном виде принято называть дампом памяти (от dump — свалка).

Вывод дампа в отладчике debug производится по команде `d` (от dump), которая имеет следующую структуру:

```
d <диапазон адресов>
```

где `<диапазон адресов>` задает начальный и конечный адреса участка памяти, дамп которого нужно выдать. Эти адреса задаются с помощью расположенных после названия сегментного регистра и разделенных пробелом внутрисегментных смещений начала и конца участка, например:

```
d ds:100 120
```

Обратите внимание! Конечный адрес участка задается только смещением, название сегментного регистра задавать вновь нельзя! Естественно, конечный адрес должен быть не меньше начального. Кроме того, конечный адрес может отсутствовать. В этом случае программа выведет на экран содержимое 128 байтов, начинающихся с заданного начального адреса.

На рис. 4.19 показаны две команды вывода дампа памяти. В первом случае запрашивается вывод дампа с адреса `ds:0020`, а во втором — с адреса `cs:0100`. Каждая строка дампа начинается с адреса выводимого участка памяти, записываемого в виде *S:D*. Далее в виде шестнадцатеричных кодов выводится содержимое 16 последовательных байтов памяти. А затем это же содержимое отображается в символьном виде с использованием кодировки ASCII. Именно такая форма вывода считается стандартной для дампов памяти. Просматривая примеры дампов памяти, легко убедиться в том, что отладчик совершенно правильно заполнил указанные в предыдущих командах `e` поля памяти.

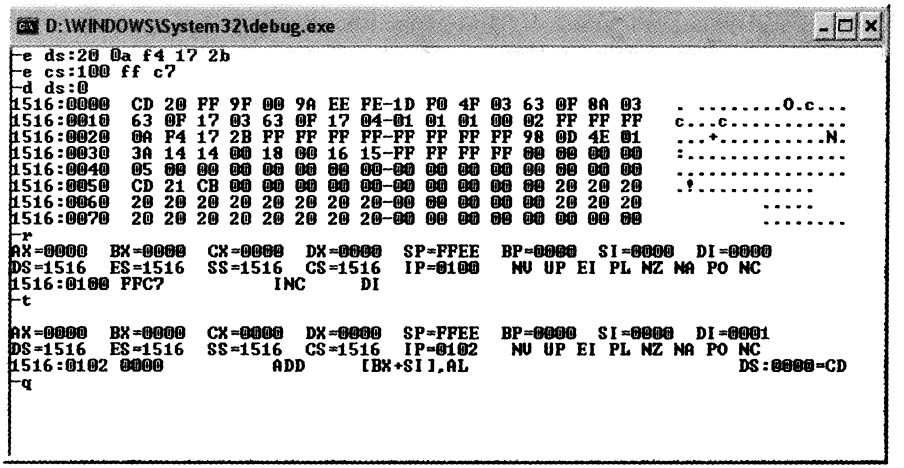


Рис. 4.20. Примеры команд отладчика debug

Для просмотра содержимого некоторого регистра процессора служит команда `r` (от register), имеющая следующий формат:

```
r <имя регистра>
```

где `<имя регистра>` — символьное название регистра. Например, команда `r cx` выводит на экран содержимое регистра `cx`.

Эта команда может использоваться без параметров, при этом на экран выводится содержимое всех регистров процессора, значения всех флажков, а также адрес, код и ассемблерный формат текущей машинной команды. Если в команде определяется адрес поля памяти, то выводится и его содержимое вместе с вычисленным эффективным адресом. Заметим, что именно в такой форме удобнее всего использовать команду `r`.

На рис. 4.20 приведен пример выполнения команды `r` без параметров. Видно, что операционная система при запуске программы загружает нулевые коды 0000_{16} во все регистры общего назначения, а также индексные и базовые регистры (кроме регистра `sp`, о котором мы будем говорить далее). Сегментные регистры содержат коды, значения которых существенно зависят от текущей ситуации, от используемой операционной системы, от имеющегося в компьютере фактического объема оперативной памяти и от многих других факторов. В данном случае во все сегментные регистры загружен код 1516_{16}.

Таблица 4.18. Обозначения состояния флажков процессора в отладчике debug

Флажок	Of	df	if	sf	zf	af	pf	Cf
Нуль	NV	UP	EI	PL	NZ	NA	PO	NC
Единица	OV	DN	DI	NG	ZR	AC	PE	CY

Далее во второй строке вывода отладчика находятся содержимое регистра `ip` и цепочка буквенных обозначений состояний флажков процессора: `NV UP EI PL`

NZ NA PO NC. Соответствие между значениями флажков и их обозначениями в отладчике приведены в табл. 4.18. Отметим, что флажки в таблице приведены в порядке их размещения в регистре флажков, а обозначения значений для флажка трассировки tf отсутствуют. В рассматриваемом примере флажки показывают, что переполнение отсутствует (of = NV = 0), цепочечные операции выполняются в направлении возрастания адресов (df = UP = 0), прерывания разрешены (if = EI = 0), результат неотрицательный (sf = PL = 0), результат ненулевой (zf = NZ = 0), дополнительного переноса нет (af = NA = 0), паритет нечетный (pf = PO = 0) и перенос отсутствует (cf = CY = 0).

В последней, третьей строке отладчик выводит полный адрес текущей команды 1516:0100, ее машинный FF C7 и ассемблерный inc di форматы.

Для выполнения программы в отладчике имеются две команды. Команда g (от go) в формате

```
g <адрес поля>
```

запускает на выполнение машинную программу в обычном (безостановочном) режиме. Выполнение начинается с команды, адрес которой задан параметром <адрес поля>. Если параметр в команде опущен, то отладчик пытается осуществить запуск программы с адреса, определяемого содержимым пары cs:ip.

Команда t (от trace — следить) в том же формате

```
t <адрес поля>
```

обеспечивает запуск программы в режиме трассировки, в котором процессор останавливается после выполнения каждой машинной команды программы, а затем выводит на экран ту же самую информацию, что и по команде r без параметров.

В нижней части рис. 4.20 показан пример выполнения команды трассировки t. Обратите внимание на содержимое регистра di и сравните его со значением, выданным по предыдущей команде просмотра регистров. Видно, что команда инкремента его содержимого уже выполнена. Далее обратите внимание на содержимое регистра ip, на адрес следующей команды 1516:0102, на ее код 0000_{16} и на ее ассемблерный формат add [dx+si],al. Видно, что в команде используется косвенная адресация с базированием и индексированием [dx+si]. Поскольку операнд команды выбирается из поля памяти, отладчик определяет его физический адрес и помещает и сам адрес, и операнд, находящийся по этому адресу, в конце последней строки вывода ds:0000 = CD.

Завершение работы отладчика осуществляется с помощью команды q, не имеющей параметров. Отметим, что закрытие окна другими способами (например, сочетанием Alt+F4) некорректно и вызывает «возражения» операционной системы.

Создание машинной программы

Изученного материала вполне хватит для того, чтобы получить общее представление о разработке программы на уровне машинного языка компьютера. С целью рассмотрения конкретного примера предположим, что анализ и построение

решения некоторой задачи привели к выводу о необходимости выполнить следующие действия.

- Записать код числа $+29_{10}$ в регистр ax.
- Увеличить это число на 1.
- Скопировать результат в регистр di.
- Записать код числа $+10_{10}$ в сегмент данных в поле со смещением из регистра di.
- Уменьшить это число на 1.
- Из содержимого регистра ax вычесть число, находящееся в выбранном поле памяти, результат поместить в это же поле памяти.
- Очистить регистр di.
- Вернуться в операционную систему.

Не следует пытаться каким-либо образом интерпретировать эти действия. Их подбор диктовался только одной целью: проиллюстрировать особенности команд различных форматов. Коды операций необходимых машинных команд находятся в табл. 4.4, а кодировки регистров — в табл. 4.9 и 4.11.

Первое действие можно выполнить с помощью команды пересылки непосредственного операнда в регистр. По табл. 4.4 находим, что эта команда имеет код операции 1011_2. Ее формат представлен на рис. 4.12, *д*: КОП|w|reg|B_1|B_2, где B_1 и B_2 — младший и возможный старший байты пересылаемого кода. В рассматриваемом случае код числа $29_{10} = 11101_2 = 1D_{16}$ нужно записать в двухбайтовый регистр ax, следовательно, параметр w должен быть равным 1. Это влечет за собой необходимость включения в команду кода непосредственного данного, занимающего два дополнительных байта 00 $1D_{16}$. По табл. 4.9 находим код регистра ax: 000_2. Таким образом, искомая машинная команда должна иметь вид КОП = 1011|w = 1|reg = 000|B_1 = 00011101|B_2 = 00000000. Исключая пояснительные обозначения, которые при дальнейшем изложении в формируемом коде команды указываться не будут, получим окончательный вид команды: $1011|1|000|00011101|00000000_2$ = = B8 1D 00_{16}.

Увеличить значение в регистре на 1 можно с помощью команды сложения с непосредственным операндом. Однако проще выполнить это действие с помощью команды инкремента, с кодом операции 01000_2. Ее формат представлен на рис. 4.12, *б*: КОП|reg. С учетом уже известного значения параметра reg = 000_2 получим машинную команду в виде $01000|000_2 = 40_{16}$.

Пересылка кода из одного места в другое осуществляется двухадресной командой с кодом операции 100010_2 и форматом вида 4.12, *г*: КОП|d|w|mod|reg|r/m. Поскольку речь идет о пересылке из регистра в регистр, то параметр mod = 11_2 Регистры двухбайтовые — значит, w = 1. При mod = 11_2 код регистра ax — 000_2, а код регистра di — 111_2. Исходя из того, что целевым регистром, то есть регистром, в который нужно поместить результат, является di, можно предложить два варианта команды. Если выбрать значение параметра d = 0, то кодировать регистр di нужно параметром r/m. Тогда команда примет вид $100010|0|1|11|000|111_2$ = $1000\ 1001\ 1100\ 0111_2$ = 89 $C7_{16}$. Если же выбрать d = 1, то кодировать di сле-

дует параметром reg. В этом случае получим: 100010|1|1|11|111|000$_2$ = 1000 1011 1111 1000$_2$ = 8B F8$_{16}$. Оба варианта команды равноценны и выполняют одно и то же действие. Для определенности выберем последний вариант.

Пересылка в поле данных, адрес которого задан в регистре di, может быть выполнена вторым вариантом команды с непосредственным операндом, формат которой приведен на рис. 4.12, *е*: КОП|W|mod = 00|КОП$_д$|r/m, при этом параметр mod всегда равен 00$_2$. Коды операции этой команды: КОП = 1100011$_2$ и КОП$_д$ = 000$_2$. В данном случае требуется использовать косвенную адресацию с индексированием через регистр di. Так как mod = 11$_2$, то кодировку параметра r/m для регистра di ищем в табл. 4.11: r/m = 101$_2$. Поскольку W = 1, код непосредственного операнда +10$_{10}$ = 1010$_2$ = 0A$_{16}$ должен занимать в команде два дополнительных байта. С учетом сказанного получаем команду в виде 1100011|1|00|000|101|00011110|00000000$_2$ = = C7 05 0A 00$_{16}$.

Декремент кода в поле памяти выполняется одноадресной командой в общей модификации с кодами КОП = 111111$_2$ и КОП$_д$ = 001$_2$ и форматом, изображенным на рис. 4.12, *в*: КОП|W|mod|КОП$_д$|r/m. Поле памяти вновь задается косвенной адресацией с индексированием через регистр di, поэтому mod = 00$_2$, а r/m = 101$_2$. Таким образом, получена машинная команда в виде 1111111|1|00|001|101$_2$ = FF 0D$_{16}$.

Требуется вычесть из содержимого поля памяти, адресуемого через регистр di, содержимое регистра ax и результат поместить назад в то же самое поле памяти. Вычитание относится к двухадресным командам с кодом операции 001010$_2$ и уже встречавшимся форматом вида КОП|d|W|mod|reg|r/m. Кодировка нужного вида косвенной адресации также уже обсуждалась: mod = 00$_2$ и r/m = 101$_2$. Так как операнд считается двухбайтовым, то W = 1. Вычитаемое находится в регистре ax, код reg которого 000$_2$. Результат нужно отправить по адресу, заданному параметром r/m, следовательно, d = 0. В итоге получим команду 001010|0|1|00|000|101$_2$ = 29 05$_{16}$.

Под **очисткой регистра** понимается запись в этот регистр нулевого значения. Такое действие можно осуществить самыми разными способами. Один из популярных вариантов состоит в организации вычитания из регистра его же содержимого. Код команды вычитания и ее формат только что рассматривались. Но в этом случае оба операнда находятся в регистрах процессора, поэтому mod = 11$_2$, а код регистра di не 101$_2$, а 111$_2$, так как он определяется не по табл. 4.11, а по табл. 4.9. Приходим к следующему виду команды: 001010|0|1|11|111|111$_2$ = 29 FF$_{16}$.

Программы, работающие под управлением современных операционных систем, принято завершать не командой остановки F4$_{16}$, а командой, возвращающей управление операционной системе, CB$_{16}$. Если команда CB$_{16}$ нарушает выравнивание на границу слова, то перед ней целесообразно поставить так называемую **пустую команду** 90$_{16}$, которая, не выполняя никаких действий, обеспечивает нужное размещение следующей за ней команды.

Для выполнения программа в машинных кодах должна быть загружена в оперативную память по адресу, задаваемому указателем cs:100, при этом содержимое

регистра cs определяется операционной системой автоматически. На рис. 4.21 изображен участок оперативной памяти, который содержит разработанную программу.

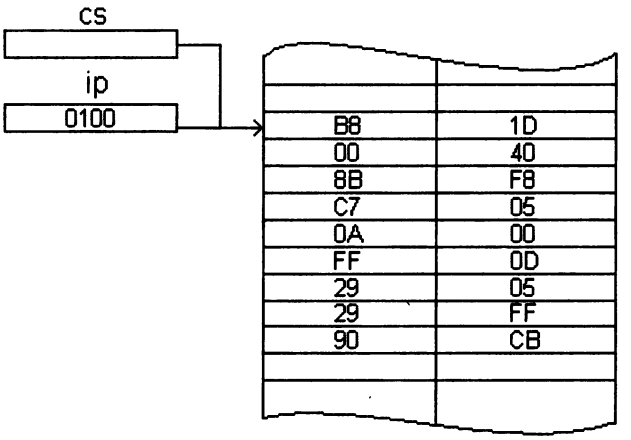


Рис. 4.21. Пример программы в машинных кодах

Правильность составленной программы должна быть подвергнута самой тщательной проверке. Программы, имеющиеся только в машинных кодах, могут быть проверены с помощью отладчика. В простейшем варианте для проведения отладки подбирается набор исходных данных, для которого несложно определить конечные результаты без выполнения программы на компьютере. Затем программа выполняется, и если результаты полностью совпадают, можно с определенной долей уверенности считать программу правильной. При их несоответствии выполнение программы анализируется шаг за шагом до первого несовпадения предполагаемых и выдаваемых компьютером результатов. Выявляются причина несовпадения, местоположение и характер ошибки. После ее исправления процедура повторяется до полного совпадения результатов. Поскольку вопросы проведения отладки не являются предметом нашего изучения, мы ограничимся только этим первоначальным ее этапом.

Если проанализировать последовательность действий, для которых составлена рассматриваемая учебная программа, можно убедиться в том, что ее результатами должны быть код числа $+29_{10} = 1D_{16}$ в регистре ax, код числа $-21_{10} = FF\ EB_{16}$ в сегменте данных в поле со смещением $001E_{16}$ и нулевой код в регистре di.

Проверяемую программу вначале нужно передать отладчику. Если она находится на дисковом носителе, то передача осуществляется с помощью не обсуждавшихся ранее команд отладчика n и l. Короткую программу можно ввести в отладчик с клавиатуры, применяя команду e.

На рис. 4.22 приведен протокол ввода и трассировки написанной ранее машинной программы. С помощью первой команды отладчика вводится код всей программы. Затем с помощью команд d cs:100 11F и d ds:0 1F просматриваются дампы двух участков оперативной памяти. Первый участок из сегмента кода

просматривается с целью проверки правильности ввода кода программы: байты памяти, начинающиеся с адреса cs:0100 и заканчивающиеся адресом cs:011F, должны содержать код программы. При необходимости с помощью дополнительных команд редактирования отдельные байты могут быть исправлены. Выводится также дамп участка сегмента данных от адреса ds:0000 до адреса ds:001F, в котором должен быть получен результат. Последующее сравнение этого дампа с дампом, полученным после выполнения программы, позволит убедиться в правильности или неправильности выполнения программы. В данном случае следует обратить внимание на байты со смещениями $001E_{16}$ и $001F_{16}$, которые должны содержать результат после выполнения программы. Заметим, что перед выполнением программы эти байты содержат код FF_{16}.

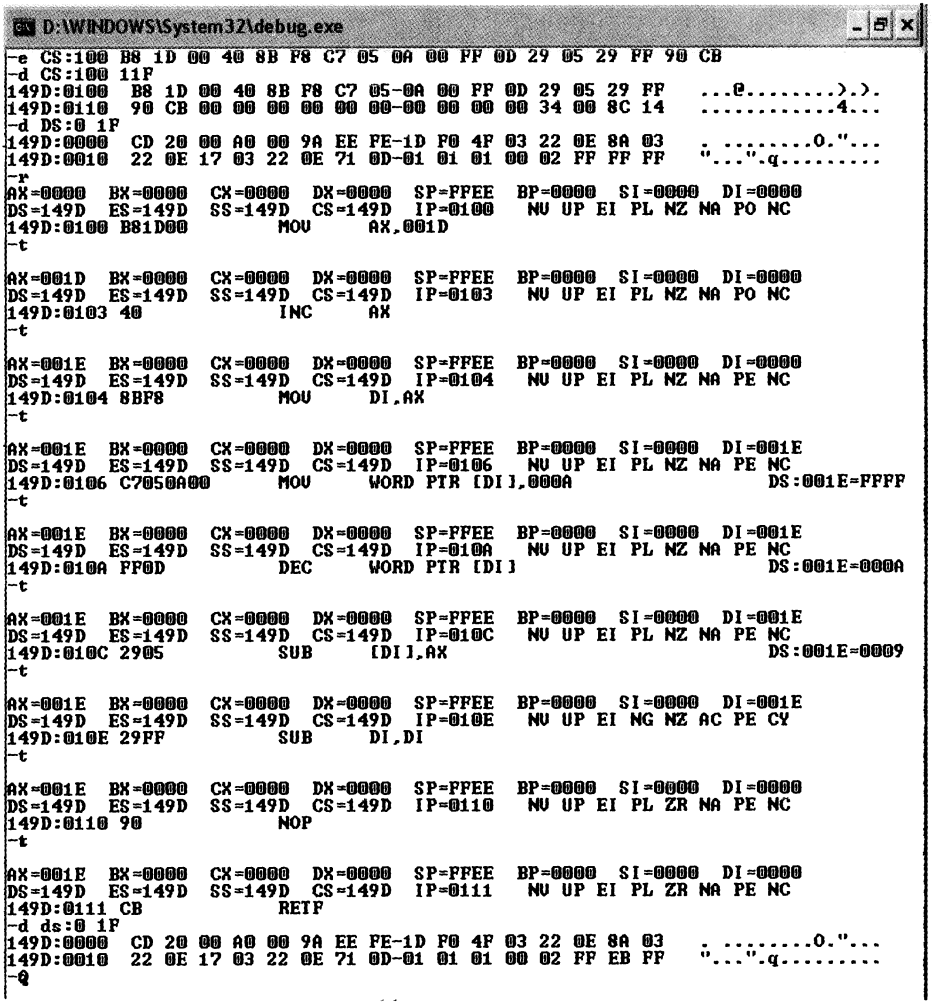


Рис. 4.22. Пример трассировки программы в машинных кодах

Следующая команда `r` обеспечивает просмотр кодов, находящихся в регистрах процессора. В нашем случае нужно обратить особое внимание на содержимое регистров `ax` и `di` — в них находятся нулевые коды 00 00_{16}, а также на регистр `ip` — он содержит код 0100_{16}, обеспечивающий выборку первой команды программы. Она приводится далее в шестнадцатеричном виде B8 1D 00_{16} и в ассемблерном формате `mov ax,001D`. В связи с этим говорят, что отладчик выполняет **дизассемблирование** программы. Напомним, что ассемблированием называется преобразование мнемокодов ассемблерного формата в машинные коды. А дизассемблирование, очевидно, выполняет обратное действие преобразования машинных команд в ассемблерный формат. Следует также внимательно проанализировать состояние флажков процессора. На рисунке видно, что все флажки установлены в исходное нулевое состояние.

Таблица 4.19. Трассировка программы

№	Выполненная команда	Результаты	Действие	Регистр ip
1	`B8 1D 00` `mov ax,001D`	`[ax]=00 1D`	Пересылка кода числа $+29_{10}$ в регистр `ax`	`[ip]=0103`
2	`40` `inc ax`	`[ax]=00 1E`	Увеличение значения в `ax` на единицу	`[ip]=0104`
3	`8B F8` `mov di,ax`	`[di]=00 1E`	Пересылка кода из регистра `ax` в регистр `di`	`[ip]=0106`
4	`C7 05 0A 00;` `mov word ptr [di]`	`ds:001E=00 0A`	Пересылка кода числа $+10_{10}$ в поле памяти	`[ip]=010A`
5	`FF 0D;` `dec word ptr [di]`	`ds:001E=00 09`	Уменьшение числа в поле памяти на единицу	`[ip]=010C`
6	`29 05;` `sub [di],ax`	`sf=NG`	Результат вычитания отрицательный	`[ip]=010E`
7	`29 FF; sub di,di`	`[di]=0000;` `zf=ZR`	Результат нулевой	`[ip]=0110`

По первой команде трассировки `t` выполняется команда B8 1D 00_{16}. В регистр `ax` заносится код 00 $1D_{16}$. Состояние всех флажков при этом остается неизменным. Кроме того, происходит увеличение значения указателя команд на длину выполненной команды: `[ip]` = 0103_{16}. По полученному адресу находится следующая команда программы 40_{16} или `inc ax`. Необходимо отчетливо понимать, что по командам `r` и `t` отладчика на экран выводится текущее состояние регистров процессора и его флажков. Но команда, которая приводится далее в той же группе строк вывода, является следующей командой трассируемой программы, она будет выполнена только по следующей команде трассировки отладчика.

В дальнейшем обсуждении хода трассировки целесообразно фиксировать внимание только на затронутых выполнением текущей команды изменившихся значениях в регистрах и полях памяти. Для наглядности результаты такого анализа

сведены в табл. 4.19. Данные приведены в шестнадцатеричных кодах. Во втором столбце находится выполненная команда программы в шестнадцатеричном виде и ассемблерном формате. В последнем столбце — содержимое регистра `ip`, которое определяет адрес следующей команды программы. Два средних столбца содержат изменившиеся элементы строк выдачи отладчика, на которые следует обратить внимание во время их анализа, и их краткое пояснение.

Результаты выполнения программы, которыми являются выданные по последней команде трассировки данные, позволяют убедиться в том, что регистры `ax` и `di` процессора содержат нужные значения. А последний вывод дампа сегмента данных показывает, что в поле памяти с указателем `ds:001E` находится код FF EB_{16}, который и должен быть в этом поле получен.

ПРИМЕЧАНИЕ

Не забывайте об обратном порядке записи кодов в оперативной памяти. Отладчик debug в дампах памяти сохраняет фактический порядок, а во всех остальных элементах вывода формирует подразумеваемый код.

Таким образом, в первом приближении можно считать, что составленная программа соответствует поставленной задаче.

Рассмотренный в данном разделе пример работы на уровне машинных кодов довольно наглядно показывает сложность этой работы. Она требует знания системы команд процессора на уровне мельчайших деталей и тщательного, весьма трудоемкого программирования, когда описание выполняемых действий производится с привлечением таких понятий, как регистры процессора и его флажки. Вероятность совершения ошибки огромная, а ее поиск, даже с привлечением отладчика, требует кропотливого, зачастую сопряженного со значительными временными затратами анализа. В общем, это малопроизводительная, весьма сложная, требующая высочайшей квалификации работа. Заметим, что перечислены далеко не все проблемы, с которыми она сопряжена.

Описанный способ работы над программой считается **низкоуровневым**. Он был характерным для начального периода развития вычислительной техники. Но довольно быстро его заменили программированием на более высоком уровне языка Ассемблер, который обеспечивает практически все доступные на низком уровне возможности доступа к аппаратным средствам компьютера, но является значительно более удобным и эффективным способом разработки программ.

Вместе с тем, даже при использовании самых современных инструментальных систем остается необходимость в критических ситуациях анализа программ и ошибок выйти на уровень битов, байтов, регистров, флажков, машинных команд и т. д. Поэтому практически все развитые инструментальные системы содержат средства доступа к этому уровню На рис. 4.23 приведен пример окна центрального процессора CPU, которое может быть вызвано в инструментальной среде Delphi командой View ▸ debug Windows ▸ CPU. Несмотря на то что окно отображает состояние процессора *32-битной* архитектуры, в нем можно обнаружить знакомые элементы. Большую часть окна (слева вверху) занимает текст программы

в шестнадцатеричном виде и ассемблерном формате. Команды делятся на группы, соответствующие помещенной над группой конструкции языка Паскаль. Слева от команд программы приведены их смещения в сегменте кодов. Справа вверху находятся два сектора, которые отображают содержимое регистров процессора и состояние его флажков. Сектор внизу слева содержит дамп сегмента кодов, а внизу справа — дамп сегмента данных. Легко увидеть, что это окно содержит ту же самую информацию, которая доступна и в отладчике. Но в отладчике для получения доступа к различным элементам нужно последовательно вводить различные его команды, в то время как в обсуждаемом окне все они видны одновременно.

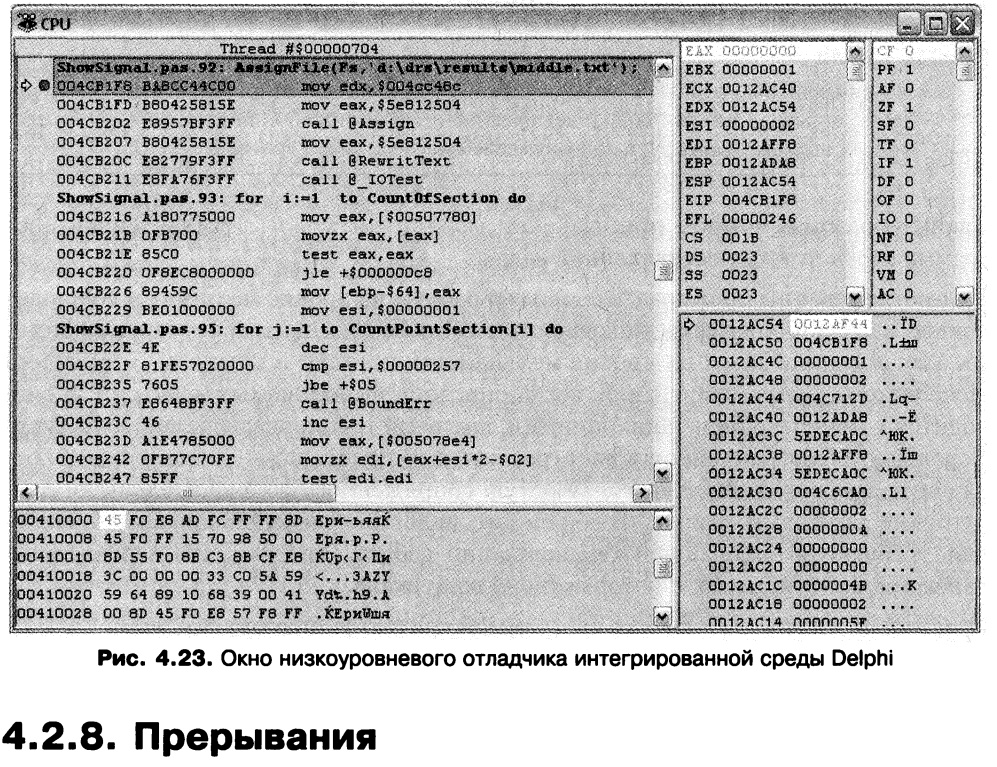


Рис. 4.23. Окно низкоуровневого отладчика интегрированной среды Delphi

4.2.8. Прерывания

Известно, что выполнение любой программы связано не только с обработкой данных, находящихся в оперативной памяти, но и с операциями обмена данными между программой и внешними устройствами компьютера. Способ взаимодействия между процессором и оперативной памятью во время выполнения программы был рассмотрен ранее. Теперь кратко обсудим взаимодействие между процессором и внешними устройствами во время выполнения включенных в программу операций обмена.

Обрабатывая данные, процессор обращается к байтам оперативной памяти, а для выполнения операций обмена процессору нужно обращаться к так называемым портам ввода/вывода, которые соединены с внешними устройствами компьютера.

На логическом уровне каждый порт представляет собой восьмибитовое запоминающее устройство, предназначенное для получения передаваемого в процессе обмена байта данных. Порт можно рассматривать как некоторое окно или канал, через который процессор передает внешнему устройству или принимает от него код одного байта. Физически порт представляет собой связанный с разъемом на корпусе или на шине компьютера **буфер памяти**, свойства которого аналогичны свойствам байта оперативной памяти. Отметим, что в связи с этим разъемы на корпусе компьютера также часто называются портами: например, порт LPT1, связанный с принтером, или порт COM, к которому может подключаться мышь или модем.

Для организации обмена в системе команд процессора i8086 предусмотрены две команды: ввода `in` (КОП = 1110010_2) и вывода `out` (КОП = 1110011_2). Во время ввода переданный с внешнего устройства байт данных помещается в связанный с этим устройством порт, а затем по команде `in` программы этот байт из порта передается в фиксированный (всегда один и тот же) регистр `al` процессора. Во время вывода, наоборот, процессор по команде `out` программы передает байт данных из регистра `al` в указанный в команде порт, откуда этот байт попадает на внешнее устройство. Два восьмибитовых порта могут объединяться для получения и передачи двухбайтового кода. В этом случае в обмене участвует регистр `ax` процессора. Еще раз отметим, что в операциях ввода/вывода участвует только регистр `al` (или `ax`), это одна из его специализированных функций.

В процессоре i8086 предусмотрено 2^{16} = 65 536 портов. Каждый порт, так же как и каждый байт оперативной памяти, имеет номер, который также считается адресом. Совокупность номеров портов образует **адресное пространство ввода/вывода** объемом 64 Кбайт, которое не зависит от адресного пространства оперативной памяти.

Выбор порта, к которому подключается то или иное внешнее устройство, может осуществляться программистом «вручную» или же производится операционной системой автоматически во время так называемого **конфигурирования** устройства. Далее номер порта, с которым связано участвующее в обмене внешнее устройство, указывается в командах ввода/вывода `in` и `out`.

Понятие прерывания

В связи с большим разнообразием внешних устройств процесс обмена в целом является довольно сложной процедурой. Он не сводится только к выполнению начинающих или завершающих обмен команд `in` или `out`. Так, например, клавиатура, передающая данные при вводе, должна каким-то образом известить процессор о том, что очередной байт принят от пользователя (после нажатия им очередной клавиши) и передан во входной порт. Аналогичным образом устройство, принимающее очередной байт данных при выводе, должно иметь возможность информировать процессор о том, что оно готово к его приему. Следовательно, нужна детально проработанная схема взаимодействия между процессором и внешними устройствами во время обмена в целом.

ВНИМАНИЕ

Здесь речь идет не об обмене данными между процессором и внешними устройствами, а о взаимодействии между процессором и внешними устройствами во время обмена.

Теоретически возможны два способа такого взаимодействия: **опрос** и **прерывание**. Опрос представляет собой периодическое прекращение текущей работы процессора и посылку им запросов каждому из внешних устройств на возможность или необходимость выполнения операции по обмену. Опрос похож на следующую бытовую ситуацию. Учитель (процессор) по очереди обходит всех учеников (внешние устройства) и у каждого спрашивает, не появились ли у него вопросы по решаемой задаче или обсуждаемой теме. Очевидным недостатком такого способа взаимодействия является то, что учитель (процессор) подавляющую часть времени будет заниматься не объяснением учебного материала (обработкой данных), а выяснением наличия у учеников (внешних устройств) вопросов.

Поэтому в компьютере обмен данными организован с помощью прерываний, одним из важнейших элементов которых является запрос на выполнение операции обмена, *посылаемый процессору со стороны внешних устройств или же выполняющейся программы*. В рассмотренной ранее аналогии это соответствует ученику, который сам привлекает внимание учителя только тогда, когда у него появится вопрос.

ВНИМАНИЕ

Под прерыванием понимается прекращение выполнения одной программы и переключение процессора на выполнение другой программы. При необходимости выполнение прерванной программы может быть возобновлено.

Чтобы более точно понять сущность прерывания, вновь рассмотрим бытовой аналог понятия. Человек сидит за столом и читает книгу (процессор выполняет некоторую программу). Раздается звонок телефона (в процессор поступает сигнал прерывания, то есть запрос, требующий, чтобы процессор обратил внимание на конкретное внешнее устройство). Человек прекращает выполнение текущего действия и переключается на выполнение другого. Он прекращает чтение, поднимает телефонную трубку и начинает разговаривать (процессор приостанавливает выполнение текущей программы и начинает выполнение другой программы, связанной с реакцией на поступивший сигнал прерывания). После завершения разговора человек кладет трубку на телефонный аппарат и возобновляет чтение книги (процессор обработал прерывание и вернулся к выполнению ранее прерванной программы). Человеку может поступить другой запрос на прерывание, например, компьютер выдаст ему сообщение о том, что поступило почтовое сообщение. В этом случае он реагирует на прерывание другим способом.

Если внимательно проанализировать поведение человека по отношению к внешнему миру, то окажется, что обычно он использует механизмы прерываний, а не механизмы опроса, хотя в ряде случаев используются и опросы. Привлекая

далее бытовые аналогии, можно заметить, что, в принципе, человек может отложить обработку прерывания или вообще не обращать внимание на поступивший запрос (почту можно прочитать в другое, более удобное время, а на телефонный звонок можно и вообще не отвечать). Отметим также, что после обработки прерывания человек может вернуться к прерванным ранее действиям. Однако ситуация, связанная с сигналом прерывания, может не позволить человеку вернуться к прерванным действиям, скажем, если поступит сигнал о возникшем в помещении пожаре. Как мы увидим в дальнейшем, аналоги всех этих ситуаций возникают и при обработке процессором запросов на прерывание.

Физически прерывание организуется с помощью сигнала IRQ (от Interrupt ReQuest — запрос прерывания), поступающего в процессор от одного из устройств компьютера. С его помощью устройство извещает процессор о том, что для дальнейшего выполнения обмена ему требуются некоторые действия процессора. Получив такой сигнал, процессор запоминает свое текущее состояние и всю необходимую для возобновления работы информацию, а затем переключается на выполнение специальной программы, которая называется **программой обработки прерывания**. В процессе ее выполнения устройство передает процессору некоторую дополнительную информацию о создавшейся во время выполнения обмена ситуации. Обычно эта информация представлена в виде так называемого **кода завершения** (функции завершения), специфического для каждого из устройств, выполняющих обмен. Код завершения показывает, правильно или неправильно, полностью или не полностью завершился обмен, требуется ли вмешательство пользователя для дальнейшего его выполнения (например, в случае отсутствия бумаги в принтере) и т. д. Код завершения может также показывать, что запрошенные действия устройство выполнить не в состоянии. В соответствии с полученной в коде завершения информацией процессор должным образом организует дальнейшую обработку прерывания. Затем он может вернуться к выполнению ранее прерванной программы.

Классификация прерываний

Чтобы эффективнее организовать обработку прерываний, введена их классификация. В частности, различают классы **внешних** и **внутренних** прерываний. Внешние прерывания происходят от внешних по отношению к процессору устройств. К ним относятся **прерывания ввода/вывода**, которые поступают в процессор от устройств, завершивших обмен и требующих в связи с этим определенной реакции процессора, а также **аппаратные прерывания**, поступающие в процессор от различных устройств компьютера в случае обнаружения сбоев в их работе.

Для обеспечения единообразия в обработке различных операций по обмену и возникающих при этом ситуаций введен класс внутренних прерываний, которые происходят внутри самого процессора. К ним относятся **программные прерывания** и **прерывания от исключительной ситуации**. Программными прерываниями считаются, например, запросы со стороны программы на начало выполнения ввода иди вывода. Кроме того, программные прерывания широко используются для обращения к разнообразным средствам автоматизации программ, предусмотренным

в операционной системе. Программные прерывания выполняются с помощью включения в программу команды `int n` (КОП = 1100110_2), где n — номер прерывания. Исключительные ситуации возникают в тех случаях, когда процессор *не в состоянии* выполнить предусмотренное в программе действие, например деление на нуль. В этом случае он прерывает выполнение программы и сообщает инструментальной среде или операционной системе о наличии исключительной ситуации, а также выдает необходимую дополнительную информацию, которая позволяет более точно определить причину ее возникновения.

Типы прерываний

В связи с большим разнообразием ситуаций, возникающих во время обмена с различными внешними устройствами, для правильной обработки прерывания кроме собственно сигнала на прерывание необходима некоторая дополнительная информация. Аналогичным образом, после стука в дверь (поступления сигнала прерывания) следует вопрос: «Кто там?», который представляет собой просьбу дать дополнительную информацию о стучащем (о прерывании). Дополнительная информация о характере прерывания передается по шине данных в процессор в виде целочисленного кода n, $0 \le n \le 255$, который называется **типом**, или **номером**, **прерывания**. Следовательно, с помощью номера n могут различаться 256 типов прерываний. Именно этот номер включается в упоминавшуюся ранее команду программного прерывания `int n`.

Далее представлены некоторые типы прерываний и их номера:

- тип 0 — деление на нуль;
- тип 1 — пошаговая работа (трассировка);
- тип 2 — катастрофическая ситуация, например отключение электропитания;
- тип 3 — прерывание из программы с помощью специальной команды INT;
- тип 4 — переполнение порядка при работе со знаковым операндом;
- тип 5 — выход индекса за границу;
- тип 6 — недействительный код операции и т. д.

Список всех используемых типов прерываний можно найти в технической документации.

Таблица векторов прерываний

Для каждого типа прерывания предусмотрена входящая в состав операционной системы специализированная программа, которую должен выполнить процессор при поступлении сигнала прерывания соответствующего типа. Адрес программы обработки прерывания представляет полный указатель длиной 4 байта. В связи с тем, что полный указатель состоит из двух компонентов — адреса сегмента и внутрисегментного смещения, — он рассматривается как двухкомпонентный вектор. Адреса всех программ обработки прерываний собраны в так называемой **таблице векторов прерываний**. Каждый четырехбайтовый элемент таблицы

содержит отдельный вектор прерывания — полный указатель программы обработки соответствующего прерывания. В этом векторе вначале располагаются два байта внутрисегментного смещения — содержимое регистра `ip`, а затем два байта адреса сегмента — содержимое регистра `cs`. Таблица векторов прерываний всегда находится в одном и том же месте оперативной памяти. Ее начальный элемент размещается в нулевом байте памяти (рис. 4.24). Соответственно следующий элемент начинается по адресу 00004_{16}, затем по адресу 00008_{16} и т. д. Таблица содержит отдельный элемент для каждого номера прерывания; следовательно, в таблице всего 256 элементов, а ее общая длина 1024 байта.

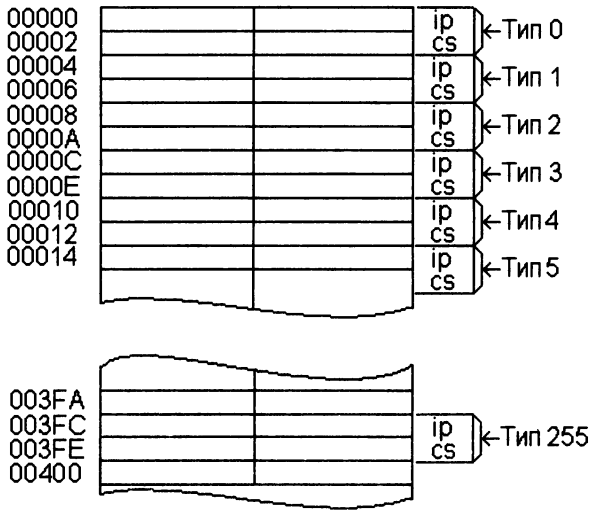


Рис. 4.24. Структура таблицы векторов прерываний

Положение указателя в таблице векторов прерываний связано с номером прерывания: адрес элемента таблицы, соответствующего прерыванию с номером n, равен $4n$. Так, для прерывания пятого типа ($n = 5$) элемент таблицы, содержащий адрес программы обработки прерывания пятого типа, находится по адресу $4 \cdot 5 = 20_{10} = 00014_{16}$.

Маскируемые и немаскируемые прерывания

В общем случае во время поступления сигнала прерывания процессор может быть занят выполнением очень важной задачи, и прерывать его работу нежелательно. Следовательно, нужно иметь возможность запрещения прерывания работы процессора. Однако и причина прерывания также может быть очень важной, требующей безотлагательного внимания процессора. Поэтому прерывания делятся на **маскируемые** и **немаскируемые**. Обработка маскируемых прерываний может быть отложена или вообще запрещена. Обработку немаскируемых прерываний отложить или запретить невозможно. В связи с этим делением для приема сигналов прерываний от внешних устройств процессор i8086 имеется два контакта

(рис. 4.25): **вход INTR** (от INTerrupt Request — запрос на прерывание) и **вход NMI** (от Not Maskable Interrupt — немаскируемое прерывание). Если ситуация такова, что обработку прерывания можно отложить, то сигнал о прерывании присылается на вход INTR. Если же ситуация требует немедленной реакции процессора (например, аналог сообщения о пожаре), то сигнал о прерывании направляется на вход NMI. Одна из ситуаций, требующих обязательной реакции процессора, — это внезапное отключение электропитания.

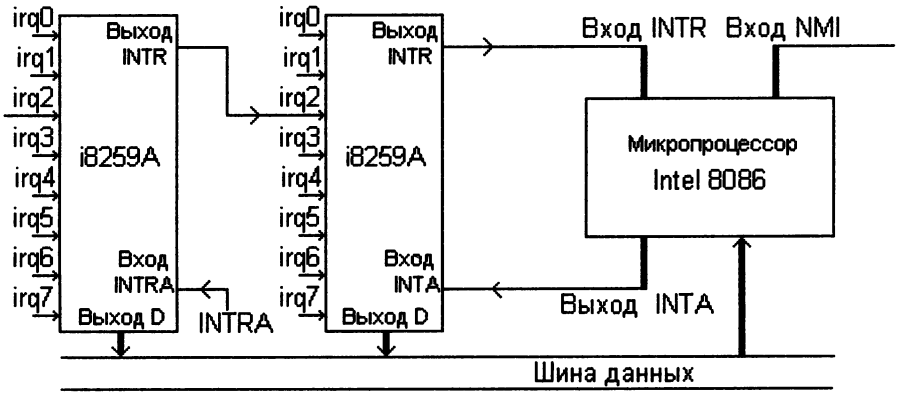


Рис. 4.25. Подсистема обработки внешних прерываний компьютера IBM PC

Действия процессора при поступлении на вход INTR сигнала прерывания зависят от состояния флажка `if`. Если `if` = 0, то прерывание не разрешено, замаскировано и запрос на прерывание процессором не обрабатывается. Если `if` = 1, то прерывание разрешено, не замаскировано и процессор начинает обработку запроса. Обработка сигнала прерывания, пришедшего на вход NMI, не зависит от состояния флажка `if`. После поступления сигнала прерывания процессор завершает выполнение текущей команды программы, фиксирует свое текущее состояние, а затем этого начинает обработку поступившего запроса. Отметим, что все прерывания, поступающие на вход NMI, имеют один и тот же номер $n = 2$ (говорят: относятся ко второму типу). Прерывания с номером $n \neq 2$ поступают на вход INTR и, следовательно, могут быть замаскированы.

Обработка прерывания

Рассмотрим немного подробнее схему взаимодействия устройств в процессе обмена. Находящаяся в программе команда ввода или вывода вызывает внутреннее программное прерывание, выполняя которое, процессор инициирует (запускает) указанное в команде устройство. Так как внешние устройства работают очень медленно, к тому же, их работа может зависеть от человека (например, в случае ввода с клавиатуры), выполнение программы, запросившей обмен, приостанавливается, а процессор переключается на выполнение какой-либо другой программы или же переходит в состояние ожидания. Завершив прием или передачу

очередного байта (или слова), запущенное процессором устройство извещает об этом процессор с помощью сигнала внешнего прерывания. Если при поступлении сигнала на вход INTR флажок `if` = 1, то есть прерывание не замаскировано, процессор обязан его обработать. Во время обработки прерывания процессор выполняет следующие действия.

- Завершает выполнение текущей команды выполняемой программы.
- Сбрасывает флажок `if` в состояние 0, запрещая на время обработки текущего прерывания поступление других внешних прерываний.
- Запоминает в стеке содержимое всех регистров (общего назначения, индексных, базовых).
- Запоминает в стеке содержимое регистра флажков.
- Запоминает в стеке адрес следующей команды выполняемой программы.
- Запрашивает у внешнего устройства тип прерывания, которое возвращается в процессор по шине данных.
- Выбирает из таблицы векторов прерываний адрес, соответствующей полученному типу программы обработки прерывания, и загружает его в регистры `cs:ip`.
- Выполняет программу обработки прерывания, после завершения которой устанавливает флажок `if` равным 1, разрешая тем самым обработку последующих прерываний.

Если возможно возобновление выполнения ранее прерванной программы, восстанавливает из стека сохраненные там данные (содержимое регистров, флажки, указатель команды) и передает ей управление, загружая указатель в пару регистров `cs:ip`.

Обычно в процессе обмена передается не один и не два байта данных. Поэтому описанная последовательность действий может выполняться циклически, до полного завершения обмена. Существуют и другие варианты организации обмена. Но в любом варианте общая схема обработки прерывания не изменяется.

Сигналы прерывания, поступившие на вход INTR при значении флажка `if` = 0, то есть замаскированные прерывания, в зависимости от ситуации либо ставятся в очередь на дальнейшую обработку, либо просто теряются.

Действия процессора при поступлении на вход NMI сигнала немаскируемого прерывания те же самые, что и при поступлении маскируемого сигнала, но не проверяется состояние флажка `if` и не запрашивается тип прерывания, так как он всегда равен 2.

Подсистема обработки внешних прерываний

Наличие в компьютере различных внешних устройств, которые одновременно могут осуществлять операции обмена, требует присутствия в его составе некоторых управляющих средств, обеспечивающих образование очереди из одновременно поступивших сигналов прерывания, передачу запросов на прерывание в процессор в установленном очередью порядке, обработку отложенных прерываний и т. д. Эти управляющие функции в компьютере IBM PC возложены на

подсистему обработки внешних прерываний (рис. 4.25), центральным звеном которой являются так называемый **контроллер прерываний**, реализованный в виде микросхемы i8259A.

ВНИМАНИЕ

Контроллером называется управляющее устройство, обеспечивающее сопряжение нескольких устройств с различной формой представления информации и отличающееся тем, что в его состав может входить независимый специализированный процессор, который выполняет управляющие функции по специальной программе.

Контроллер, по сути, является специализированным на выполнении некоторых узких функций процессором. Именно в связи с наличием в составе компьютера таких специализированных процессоров выполняющий обработку данных основной процессор обозначают CPU (от central processor unit) и называют центральным.

Отметим, что подсистема прерываний и контроллер i8259A, разработанные для компьютера на базе процессора i8086, без существенных изменений используются и в последующих моделях компьютера IBM PC.

Основными функциями контроллера прерываний i8259A являются:

- прием запросов на прерывание от внешних источников;
- формирование входного сигнала для процессора и передача его на вход INTR;
- формирование номера прерывания и передача его на шину данных;
- запрещение прерываний с определенными номерами;
- организация приоритетной обработки прерываний.

Контроллер прерываний имеет восемь входов, обозначенных на рисунке буквами irq с номером. Каждый из входов подсоединен к определенному устройству компьютера. Например, irq0 — вход таймера, irq1 — вход клавиатуры, irq6 — вход гибкого диска и т. д. Поскольку к компьютеру может быть подключено более восьми внешних устройств, в состав подсистемы может быть включено несколько соединяемых последовательно контроллеров. При этом выход INTR у первого в последовательности контроллера соединяется с входом INTR процессора, а выход INTR каждого следующего контроллера подсоединяется ко входу irq2 предыдущего (рис. 4.25).

Кроме восьми основных входов каждый контроллер i8259A имеет уже упоминавшийся выход INTR, по которому один из поступивших в контроллер сигналов прерывания передается в процессор на его вход INTR, а также выход INTA (от INTerrupt Answer — ответ прерывания), по которому процессор возвращает в контроллер сигнал о готовности обработать прерывание. Кроме того, контроллер имеет восемь выходов, обозначенных на схеме буквой D, по которым на шину передается байт, содержащий номер прерывания.

Немного упрощенная схема работы подсистемы прерываний выглядит следующим образом. Из поступивших на входы irq сигналов прерываний по определенным, заданным в его программе правилам контроллер формирует очередь. Затем за-

прос от устройства с наивысшим приоритетом (первого в очереди) передается на вход процессора INTR. Если флажок `if` = 1 и прерывания разрешены, процессор завершает выполнение текущей команды и по описанной ранее схеме готовится к прерыванию. Далее процессор устанавливает флажок `if` в нулевое состояние, запрещая тем самым выборку из очереди следующего запроса на прерывание, и выдает сигнал о готовности обработать текущее прерывание на свой выход INTA. Получив этот сигнал, контроллер передает процессору по шине данных номер прерывания, после чего процессор определяет местоположение в таблице векторов прерываний адреса программы обработки прерывания, обращается в оперативную память и загружает затем этот адрес в пару регистров `cs:ip`. После этого выбранная таким образом программа обработки прерывания выполняется стандартным образом. Завершив ее выполнение, процессор вновь устанавливает разрешающее прерывания значение флажка `if`, и из очереди выбирается следующий запрос на прерывание (если он есть), который обрабатывается по точно такой же схеме.

4.2.9. Особенности 32-битовых процессоров Intel

Рассматриваемая 16-битовая архитектура процессора i8086 была развита и усовершенствована в следующей популярной модели i80286. Все последующие модели этого семейства имеют длину машинного слова 32 бита и выше. Изучение 32-битовой архитектуры на таком же уровне подробности, как и 16-битовой, превышает возможности настоящего учебника. Поскольку общие принципы программирования на машинном уровне при переходе от 16-битовой к 32-битовой архитектуре не изменяются, в этом разделе рассматриваются только важнейшие особенности 32-битовой архитектуры.

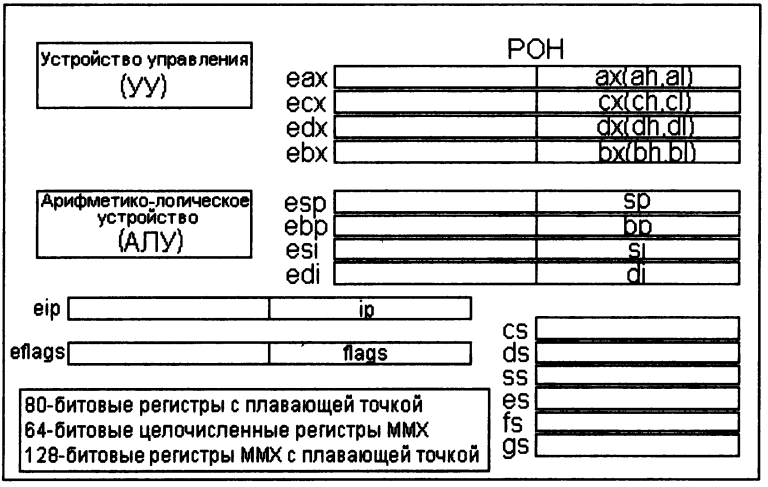


Рис. 4.26. Программная модель 32-битового процессора Intel

На рис. 4.26 изображена программная модель 32-битовых процессоров семейства Intel. Все регистры процессора, за исключением сегментных, 32-битовые, что

отражается добавлением буквы e (от extended — расширенный) в начало названий регистров. В каждом четырехбайтовом регистре процессора можно обращаться к двум младшим байтам. Таким образом, в программах можно использовать регистры eax, ecx, edx и ebx для хранения четырехбайтовых, регистры ax, cx, dx и bx — для двухбайтовых, а регистры ah, ch, dh, bh, al, cl, dl и bl — для однобайтовых кодов. У четырехбайтовых указательных esp, ebp и индексных esi и edi регистров доступны только их младшие двухбайтовые половины sp, bp, si, di.

16-битовые и 8-битовые регистры являются полными эквивалентами одноименных регистров процессора i8086. А 32-битовые регистры используются точно таким же способом, что и регистры меньшей разрядности. Например, команда `inc edi` задает увеличение на единицу четырехбайтового кода из регистра edi, а команда `add eax,ecx` определяет сложение четырехбайтовых кодов из регистров eax и ecx.

Сегментные регистры остались 16-битовыми. К имевшимся в процессоре i8086 четырем сегментным регистрам добавились еще два с названиями gs и fs, которые играют роль, аналогичную роли регистра es. То есть они используются для хранения адресов дополнительных сегментов памяти, в которых можно размещать коды данных, если объема адресуемого регистром ds сегмента не хватает.

В регистр флагов eflags добавлено 2 новых флажка состояния и 3 новых флажка управления, которые обеспечивают работу процессора в новых условиях. В 32-битовых процессорах используются адресные шины разрядности длиной 32 и 36 битов, что обеспечивает объем адресного пространства 4 или 64 Гбайт.

Обобщая аппаратные особенности 32-битовых процессоров, получим следующий перечень их программных ресурсов:

- регистры общего назначения eax/ax/ah/al, ecx/cx/ch/cl, edx/dx/dh/dl и ebx/bx/ bh/bl длиной 32/16/8/8 битов соответственно;
- индексные esi/si, edi/di и указательные esp/sp, ebp/bp регистры длиной 32/16 битов соответственно;
- сегментные регистры cs, ss, ds, es, fs, gs длиной 16 битов;
- регистр флагов eflags/flags длиной 32/16 битов соответственно;
- указатель адреса eip/ip длиной 32/16 битов соответственно;
- адресное пространство оперативной памяти объемом до 4–64 Гбайт в зависимости от разрядности адресной шины.

В связи с увеличением до 4 байтов длины машинного слова появились соответствующие форматы данных и команд, а максимальная длина команды увеличилась до 15 байтов. Кроме того, введены два новых префикса, с помощью которых осуществляется переключение в командах между 16- и 32-битовыми адресами и операндами.

32-битовые процессоры могут работать по крайней мере в двух режимах: *реальном* и *защищенном* (*виртуальном*). В реальном режиме все процессоры семейства являются аналогами процессора i8086, и любой 32-битовый процессор работает как очень быстрый 16-битовый, так как все новые возможности в этом режиме

отключаются. Реальный режим предусмотрен для сохранения программной совместимости с предшествующими моделями процессоров, для которых было разработано множество прикладных программ.

Защищенный режим предназначен для организации многопрограммной работы компьютера, которая является одним из важнейших способов увеличения его производительности. Этот режим в семействе Intel впервые появился еще у 16-битового процессора i80286. В защищенном режиме 32-битовых процессоров доступны все новые форматы данных и команд, связанные с увеличением длины машинного слова.

Сразу после включения компьютера он работает в реальном режиме, выполняя тестирование аппаратуры. После окончания загрузки операционная система формирует все структуры, необходимые для работы в ее основном, защищенном режиме, а затем переключается в этот режим функционирования.

4.3. Элементы Ассемблера процессора Intel

Дальнейшее обсуждение материала, касающегося организации линейных программ, а также ветвлений и циклов, мы будем вести не на уровне машинного языка, а с применением правил, принятых в языке Ассемблер. Это позволит избежать рассмотрения ненужных технических деталей машинного кодирования и сосредоточиться на принципиальной стороне изучаемых вопросов. По этой причине необходимо кратко ознакомиться с принятыми в этом языке правилами записи программ.

ВНИМАНИЕ

Язык Ассемблер представляет собой совокупность правил записи программ с помощью мнемокодов машинных команд.

Ассемблер относится к машинно-зависимым (машинно-ориентированным) языкам низкого уровня. Это значит, что, во-первых, язык зависит от модели компьютера, а во-вторых, из Ассемблера доступны все аппаратные возможности уровня машинных кодов, чего практически лишены языки высокого уровня — проблемно-ориентированные, функциональные и прочие языки. С другой стороны, в Ассемблере программист оперирует не громоздкими двоичными или шестнадцатеричными кодами машинных команд, а специально подобранными словами — мнемокодами, которые близки к используемым человеком в естественных языках, что существенно облегчает как разработку, так и восприятие программ.

Написанная с применением мнемокодов программа не может быть непосредственно выполнена процессором. Но она может быть автоматически преобразована (переведена, оттранслирована, ассемблирована) на уровень машинных кодов, на котором она уже может быть выполнена процессором. Такой автоматический

перевод осуществляет специализированная **программа-переводчик**, которую так же, как и язык, принято называть ассемблером.

В связи с автоматическим характером перевода программ на машинный язык отметим, что, кроме записанных с помощью мнемокодов машинных команд, определяющих требуемый порядок обработки данных, программа на Ассемблере должна содержать некоторую дополнительную информацию, например, сведения о длине полей, выделяемых для хранения промежуточных данных. Такого рода информация используется для правильной интерпретации команд программы во время ассемблирования.

Основной структурной единицей программы на языке Ассемблер является **оператор**. Операторы ассемблера делятся на **команды** и **директивы**. Команды языка Ассемблер соответствуют машинным командам процессора, а директивы ассемблера несут дополнительную информацию об используемых в программе данных, особенностях ее структуры, режимах трансляции, выполнения и т. д. В целом программа на Ассемблере представляет собой конкретную последовательность операторов, каждый из которых обычно занимает отдельную строку текста.

4.3.1. Структура оператора в языке Ассемблер

Как команды, так и директивы Ассемблера имеют одну и ту же структуру:

```
[метка] мнемокод [операнды] [; комментарий]
```

Любой заключенный в квадратные скобки элемент оператора может в нем отсутствовать. Единственным обязательным элементом является мнемокод. Каждый элемент оператора должен отделяться от другого элемента не менее чем одним пробелом. Комментарий может занимать всю строку (если метка, мнемокод и операнды отсутствуют), но в этом случае начинающая любой комментарий точка с запятой должна находиться в первой позиции строки.

Метка представляет собой символьное (то есть образованное из символов некоторого алфавита) название оператора программы. Обычно метка является буквенно-цифровым обозначением адреса поля памяти, в котором размещается код данного или код команды программы.

Метка может состоять из букв латинского алфавита (больших и маленьких), цифр и некоторых символов, которые в языке Ассемблер принято относить к специальным: вопросительный знак (`?`), точка (`.`), которая может находиться только в начале метки, знак подчеркивания (`_`); символ «AT» (`@`) и знак доллара (`$`). Имеется ряд зарезервированных имен, которые не могут использоваться в качестве меток. Это мнемокоды машинных команд (`inc`, `mov` и т. д.), названия регистров процессора (`ax`, `ah`, `al` и т. д.) и некоторые другие слова и сочетания символов (`far`, `near` и т. д.), которые используются для обеспечения возможности автоматического перевода программы с Ассемблера на машинный язык, а также во вспомогательных целях (формирование более эффективной программы, вывод заголовка листинга и т. д.). Общая длина метки не может быть больше, чем 31 символ. В остальном метка может выбираться произвольным образом,

однако так же, как и на языках высокого уровня, желательно использовать осмысленные имена.

С понятием мнемокода мы уже сталкивались — это сокращенное или полное название действия (машинной команды), операнда или директивы. Обычно мнемокод состоит из 2–7 букв и/или цифр.

Операнды операторов представляют собой двоичные, восьмеричные, десятичные или шестнадцатеричные коды или символьные обозначения. Набор операндов, а также способы их записи существенным образом зависят от оператора, в котором они используются.

На рис. 4.27, *б* изображен фрагмент программы, который содержит как директивы, так и команды. В первой и последней строках фрагмента находятся директивы, которые имеют метки. В остальных строках находятся команды, не имеющие меток. Обратите внимание на принятый способ записи операторов программы в языке Ассемблер. В принципе, не существует никаких ограничений на размещение элементов оператора в строке, кроме разделения двух соседних элементов оператора хотя бы одним пробелом. Но программисты предпочитают начинать (выравнивать) одинаковые элементы операторов на одних и те же позициях строк так, чтобы они образовывали визуально легко воспринимаемые при чтении и анализе программы столбцы.

4.3.2. Директивы сегментации программы

Программа на языке Ассемблер состоит из одного или нескольких участков, которые принято называть **сегментами программы**. Обычно выделяются сегменты для описания используемых в программе констант, полей памяти и кодов команд, задающих действия по обработке данных. Причем для каждого вида кодов выделяются отдельные сегменты. В этом смысле сегменты программы на языке Ассемблер похожи на разделы программы на языке Паскаль.

Сегменты программы, в которых сосредоточены коды команд, принято называть **программными сегментами кодов**. А ее сегменты, содержащие описания используемых в программе полей и констант, называются **программными сегментами данных**. Сегмент программы, образуемый для работы со стеком, называется **программным сегментом стека**.

ВНИМАНИЕ

Не следует путать сегмент программы и сегмент оперативной памяти. Сегмент программы — это часть текста программы, то есть оформленная по специальным правилам последовательность операторов на языке Ассемблер, а сегмент оперативной памяти — это некоторый ее участок, фрагмент аппаратуры компьютера.

Для объявления некоторого участка программы сегментом используются две директивы сегментации, `segment` и `ends` (от end segment), которыми с двух сторон ограничивается этот участок. Директива `segment` записывается в начале такого участка, а директива `ends` — в его конце (рис. 4.27, *а*).

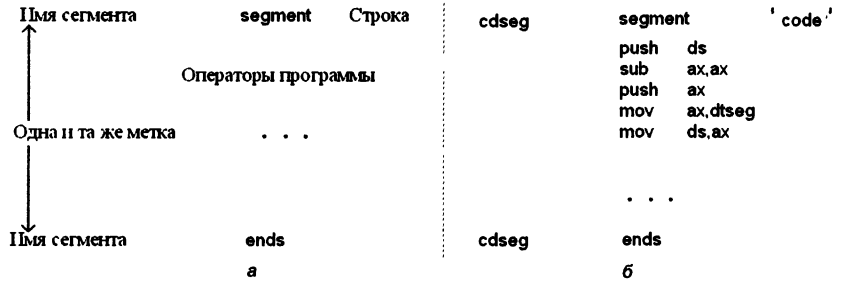


Рис. 4.27. Программный сегмент: *а* — схема организации; *б* — фрагмент

Директивы `segment` и `ends` обязаны иметь *одну и ту же метку*, по которой опознаются ограничивающие сегмент директивы сегментации. Кроме того, директива `segment` может иметь несколько параметров, из которых на рис. 4.27, *а* показан только один параметр `строка`, служащий для визуального контроля за назначением сегмента, а также для некоторых других целей. На рис. 4.27, *б* приведен пример, содержащий фрагмент программного сегмента. Директивы `segment` и `ends` обозначены меткой `cdseg`. Параметром директивы `segment` является строка `'code'`, которая часто выбирается для характеристики сегментов кода программы, содержащих мнемокоды машинных команд.

Ранее было сказано, что для выполнения любой программе *предоставляется до четырех сегментов оперативной памяти*. Далее выяснилось, что программа может состоять из произвольного количества программных сегментов. Теперь следует понять как соотносятся сегменты оперативной памяти и программные сегменты, о которых идет речь в этом разделе.

ВНИМАНИЕ

Программные сегменты размещаются в указанных программистом сегментах оперативной памяти.

Роль, закрепленная за тем или иным программным сегментом, известна только программисту, разрабатывающему эту программу. Программист произвольным образом разбивает программу на любое количество сегментов и распределяет программные конструкции между ними. Соглашение, по которому директивы определения данных сосредоточены в программном сегменте данных, *не является обязательным*. Это удобный и общепринятый способ структуризации программ, которому рекомендуется следовать. Транслирующей программе о выборе, сделанном программистом, ничего не известно.

Чтобы сообщить ассемблеру, какие программные сегменты в каких сегментах памяти следует разместить, программист может использовать директиву `assume` (принимать, присваивать), которая имеет следующий формат:

```
assume имя регистра: метка сегмента {, имя регистра: метка сегмента}
```

В соответствии с общепринятыми соглашениями конструкция, заключенная в фигурные скобки, может повторяться нуль или любое большее количество раз.

Поэтому можно сказать, что директива assume имеет один или несколько параметров, и если параметров несколько, то они отделяются друг от друга запятой.

Все параметры директивы имеют одинаковую структуру и играют одну и ту же роль. Каждый из них служит для задания связи между программным сегментом и выбранным для его размещения сегментом памяти. Параметр состоит из разделенных двоеточием названия сегментного регистра (имя регистра), с помощью которого фиксируется нужный сегмент памяти, и метки директивы сегментации (метка сегмента), которая определяет программный сегмент. Например, с помощью директивы

```
assume cs:cdsg
```

указывается, что программный сегмент с меткой cdsg нужно разместить в сегменте кода оперативной памяти, а директивой

```
assume cs:cdsg, ds:dtsg, ss:stsg
```

задается размещение трех программных сегментов с метками cdsg, dtsg, stsg на трех сегментах памяти — кода (сегментный регистр cs), данных (сегментный регистр ds) и стека (сегментный регистр ss) соответственно.

Директива assume должна быть хотя бы один раз включена в один из программных сегментов. Каждый программный сегмент, выделенный в программе, должен быть представлен в этой директиве своим параметром. Если какой-либо сегмент памяти программистом не используется, то вместо метки программного сегмента в параметре директивы можно указать слово nothing или же вообще опустить такой параметр.

Операционная система во время подготовки программы к загрузке в оперативную память выделяет под ее сегменты необходимые участки памяти. При этом адреса сегментов кода и стека автоматически загружаются в соответствующие сегментные регистры cs и ss. Затем в эти сегменты памяти записываются программные сегменты, метки которых сопоставлены этим сегментным регистрам директивой assume. Таким образом, адреса, находящиеся в регистрах cs и ss, становятся значениями меток загруженных в них программных сегментов. В условиях последнего примера метка cdsg инициализируется адресом, загруженным в регистр cs, а метка stsg — адресом из регистра ss.

ВНИМАНИЕ

Процедуру подготовки, во время которой объекты программы (метки, переменные, регистры и т. д.) получают начальные значения, принято называть инициализацией (от лат initialis — первоначальный).

В связи с тем, что операционная система использует регистр ds в процессе подготовки программы к загрузке, она не может самостоятельно загрузить в него нужный адрес и установить связь между программным сегментом и сегментом памяти. А регистр дополнительного сегмента памяти es автоматически не инициализируется операционной системой, потому что программисты не всегда используют этот сегмент памяти. Поэтому инициализация указанных регистров

должна предусматриваться программистом в разрабатываемой программе. Операционная система, используя адреса участков памяти, находящиеся в регистрах cs и ss, записывает в память соответствующие сегменты программы. Вслед за ними она размещает программные сегменты данных, если они есть. Адреса полей памяти, в которые при этом попадут их начальные конструкции, автоматически становятся значениями меток этих программных сегментов. Так метки сегментов данных получают свои значения. Остается только загрузить значения полученных адресов в соответствующие сегментные регистры. Эта процедура выполняется по специальным правилам, которые обсуждаются в дальнейшем. Пример последовательности команд, реализующих такую загрузку для регистра ds, показан на рис. 4.27, *б*.

4.3.3. Директивы определения данных

Директивы определения данных служат для резервирования в оперативной памяти так называемых **рабочих полей**, используемых для хранения исходных и промежуточных данных, а также результатов вычислений. Кроме того, с помощью этих директив резервируются поля памяти, которые содержат коды различных используемых в программе констант. Директивы определения данных в программе на языке Ассемблер во многом соответствуют описаниям переменных и констант в языках высокого уровня. Различия между указанными конструкциями мы выясним позднее. Отметим только, что директивы определения данных могут находиться в любом месте программы, в любом программном сегменте.

В стандартном Ассемблере для процессоров Intel предусмотрено пять различных директив определения данных, с помощью которых резервируются поля различной длины:

- db (от define byte — определение байта) — резервируется 1 байт;
- dw (от define word — определение слова) — резервируется 2 байта;
- dd (от define double — определение двойного) — резервируется 4 байта;
- dq (от define quarto — определение учетверенного) — резервируется 8 байт;
- dt (от define ten — определение десяти) — резервируется 10 байт.

Все директивы определения данных обязаны иметь хотя бы один операнд — параметр, который полностью определяет описываемые поля или данные. В качестве параметра директив могут использоваться следующие конструкции:

- знак вопроса (?), который рассматривается как неопределенное значение. С его помощью резервируется рабочее поле без заполнения каким-либо кодом;
- числовая константа в двоичной, восьмеричной, десятичной или шестнадцатеричной системах счисления;
- строка символов, заключенная в апострофы. Для хранения символов этой строки выделяется столько байтов, сколько символов она содержит;
- список, содержащий произвольное количество перечисленных ранее отделяемых друг от друга запятой элементов.

Примеры директив определения данных:

```
N     dw ?              ;поле для ввода количества слагаемых N
cst1  db 25             ;байт с кодом числа 25
inv   db 'Введите целое число'        ;строка текста
      db ?,25,'Введите целое число'   ;список
```

В первой строке директивой dw, которая имеет метку N, резервируется двухбайтовое рабочее поле, в которое впоследствии предполагается ввести значение переменной N. Во второй строке директивой db с меткой cst1 определяется константа 25, код которой занимает один байт памяти. Несмотря на то что в третьей строке используется директива db, формально резервирующая один байт памяти, параметр этой директивы — строка 'Введите целое число' — заставляет ассемблер выделять по одному байту на каждый символ строки. Таким образом, третья директива резервирует поле длиной 19 байтов, содержащее коды символов всей строки. В последней, четвертой строке приведен пример объединения в одной директиве нескольких определений полей и констант.

С учетом сказанного о соответствии директив определения данных Ассемблера и описаний в языках высокого уровня можно заметить, что приведенные ранее примеры директив Ассемблера соответствуют следующим конструкциям Паскаля:

```
const cst1=25;inv='Введите целое число';
var N:integer;
```

При этом директивы, резервирующие рабочие поля с помощью параметра ?, соответствуют разделу переменных, а остальные директивы — разделу констант Паскаля.

Для сокращения записи последовательностей одинаковых элементов списков можно использовать так называемый **повторитель** вида n dup(список), где n — количество повторений списка указанного в скобках, dup (от duplicate — дубликат) — ключевое зарезервированное слово, например :

```
5dup(8)       ;эквивалент списка:8,8,8,8,8
10dup('*')    ;эквивалент строки:'**********'
3dup(8,'*')   ;эквивалент списка:8,'*',8,'*',8,'*'.
```

Отметим, что повторители могут использоваться как элементы списка и вкладываться друг в друга.

При записи параметров директив определения данных входящие в них двоичные константы могут содержать только цифры 0 и 1, а в их конце должна находиться латинская буква b, например: 1000101001b. Восьмеричные константы не должны содержать цифр 8 и 9, а заканчиваться они должны буквой q: 754301q. Шестнадцатеричные константы должны заканчиваться буквой h. Если шестнадцатеричное число начинается с любой цифры от A до F, то перед числом нужно записывать цифру 0, например: 14FABh, 0FBh. В конце десятичных констант может находиться буква d, но ее можно и не указывать. Целые числа в любой системе счисления записываются по правилам, принятым в языке Паскаль. Вещественные

константы записываются только в десятичной системе счисления и также по принятым в Паскале правилам.

При выделении полей по директивам определения данных память под объекты резервируется в смежных участках, без пропусков. Во время ассемблирования программы для этого используется так называемый счетчик адреса. В начале каждой программной секции он устанавливается в нуль. После выделения поля для каждого элемента данных счетчик увеличивается на соответствующее длине элемента количество байтов. Аналогичным образом счетчик адреса работает и при формировании из команд ассемблера кодов машинных команд. Счетчик адреса может трактоваться как содержимое регистров `ip`, `bx`... в парах `cs:ip`, `ds:bx`..., содержащих полные указатели на адресуемое поле.

После определения данных и полей в директивах определения данных их метки могут использоваться в командах программы. Пример такого использования приведен в листинге 4.1.

Листинг 4.1. Фрагменты программных сегментов

```
dtsg    segment    'data'        ; начало сегмента данных
p1      dw         -8            ; определение константы
con     dw         15            ; определение константы
        ...                      ; пропущенный фрагмент сегмента
dtsg    ends                     ; конец сегмента данных
stsg    segment    'stack'       ; начало сегмента стека
        dw         512 dup(?)    ; выделение памяти под стек
stsg    ends                     ; конец сегмента стека
cdsg    segment    'code'        ; начало сегмента кода
        assume cs:cdsg,ds:dtsg,ss:stsg,es:nothing
        ...                      ; пропущенный фрагмент сегмента
        mov        ax,con1       ; использование константы
        ...                      ; пропущенный фрагмент сегмента
cdsg    ends                     ; конец сегмента кода
```

В этом примере показаны фрагменты двух программных сегментов и один сегмент целиком. В первом сегменте, ограниченном директивами сегментации с метками `dtsg`, сгруппированы директивы определения данных. В частности, в этом сегменте находится директива, определяющая константу `con1` со значением 15_{10}.

Второй сегмент, ограниченный директивами с метками `stsg`, служит для выделения 512 слов оперативной памяти (1 Кбайт) для размещения стека. Для такого выделения применена директива с повторителем `512 dup(?)`. Так как под организацию стека выделяется целый сегмент оперативной памяти, программист может выделить для него столько места, столько посчитает нужным (естественно, в пределах 64 Кбайт).

Третий сегмент, с меткой `cdseg`, содержит мнемокоды машинных команд. В его начале находится директива `assume`, устанавливающая связь между программными сегментами и сегментами памяти. В одной из команд сегмента используется описанная ранее константа `con1`.

Отметим, что при написании программы имеется определенная гибкость в выборе средств для ее реализации. Так, вместо команды `mov ax,con1`, которая использует ранее определенную метку константы `con1`, можно использовать команду с непосредственным операндом: `mov ax,15`. В связи с этим возникают вопросы: в чем разница между предложенными вариантами и какой из них лучше? Для получения ответа на эти вопросы проанализируем форматы сравниваемых команд. Первый вариант, `mov ax,con1`, очевидно, относится к двухадресным командам, так как метка `con1` представляет собой символическое обозначение адреса поля, зарезервированного в сегменте данных. Как мы уже знаем, при выполнении такой команды потребуется дополнительное обращение в оперативную память за значением операнда, находящимся в этом поле. Второй вариант, `mov ax,15`, относится к командам с непосредственным операндом в регистровой модификации, и дополнительное обращение в оперативную память не требуется. Отсюда следует уже очевидный вывод: второй вариант команды пересылки эффективнее, чем первый.

Итак, если есть возможность указать в команде непосредственный операнд, то ее желательно использовать, так как по сравнению с резервированием места в сегменте данных или в любом другом сегменте в результате ассемблирования такой команды образуется более эффективный машинный код. Отметим, что непосредственные операнды могут задаваться в командах пересылки, сравнения, сложения, вычитания и т. д.

4.3.4. Команды Ассемблера

Команды Ассемблера служат для задания действий по обработке данных, которые должен выполнить процессор, то есть они являются эквивалентами машинных команд. Количество операндов в командах Ассемблера в точности равно количеству адресов в соответствующих им машинных командах. Если операндов в команде несколько, они отделяются друг от друга запятой. В Ассемблере действует правило, в соответствии с которым результат *всегда* записывается в операнд, стоящий в команде на первом месте.

Точно так же, как и в машинных командах, операнды команд Ассемблера могут быть непосредственными, регистровыми (`ax`, `bx` и т. д.) или адресами полей памяти в прямой или различных вариантах косвенной адресации. Способы записи непосредственных операндов, обозначения регистров и способы задания прямых и косвенных адресов памяти рассматривались ранее, при обсуждении форматов машинных команд. Поэтому без подробного объяснения приведем некоторые примеры команд с различными способами адресации:

```
mov   ax,25       ;регистровая и непосредственная адресация
inc   p1          ;прямая адресация
inc   [bx]        ;косвенная адресация, базирование
inc   [bp]        ;косвенная адресация, базирование
inc   [si]        ;косвенная адресация, индексирование
inc   [di]        ;косвенная адресация, индексирование
```

```
mov   ax,[25]      ;прямая адресация, смещение в команде
inc   [bx][si]     ;базирование и индексирование
inc   v[bp][di]    ;базирование, индексирование и смещение
```

Отметим, что использованные в некоторых примерах метки `p1` и `v` обязательно должны быть описаны где-нибудь в программе с помощью директив определения данных, иначе программа не будет оттранслирована.

Для определения внутрисегментного смещения могут использоваться выражения, в которые входят арифметические операции сложения, вычитания и умножения. Например, операнд в команде `inc p1+2` является таким выражением. Важно понимать, что операнды в Ассемблере, в отличие от операндов, скажем, в Паскале, соответствуют не значениям, над которыми выполняется действие, а адресам памяти. Следовательно, действие выполняется не над значением, а над адресом. Пусть `p1` определено в сегменте, приведенном в листинге 4.1. Вспоминая о том, что метки в Ассемблере — это символические обозначения адресов памяти, приходим к выводу: `p1+2` — это адрес поля, которое имеет внутрисегментное смещение на 2 большее, чем смещение для поля `p1`. Учитывая, что в определении поля `p1` в листинге 4.1 его длина равна 2, получаем, что `p1+2` адресует поле с меткой `con1` и, следовательно, результатом выполнения команды инкремента `inc p1+2` является значение `16` в поле `con1`, — в то время как значение такого же выражения `p1+2` в языке Паскаль равно –6, поскольку значение `p1` по определению равно –8. Именно в этом состоит самое существенное отличие выражений в Ассемблере от выражений в языках высокого уровня.

Анализируя приведенные примеры, можно отметить, что при записи команд на Ассемблере не приходится заботиться о параметрах команд, таких как `W`, `d`, `mod`, `r/m`, которые с достаточным трудом подбирались при разработке программы на уровне машинных команд. Ассемблер формирует их автоматически по остальным элементам, имеющимся в команде или в других частях программы.

При выполнении большинства команд Ассемблера одновременно с получением результата формируются соответствующие значения флажков процессора `cf`, `pf`, `af`, `sf`, `zf` и `of`. Анализ этих значений позволяет организовать необходимую логику обработки данных, проверить принадлежность значения заданному диапазону и т. д.

4.3.5. Пересылка данных

Под **пересылкой данных** понимаются копирование кода, находящегося в команде пересылки (непосредственный операнд), в регистре процессора или в поле оперативной памяти, и последующая его запись в регистр или поле памяти. Адрес, по которому находится копируемый код, считается **источником**, а адрес, по которому размещается копия, считается **приемником**. Таким образом, источником пересылки могут быть выполняемая команда, регистр или поле памяти, а приемником — только регистр или поле памяти.

Собственно говоря, это действие следовало бы назвать **копированием**, а не пересылкой. Если понимать термин «пересылка» в его первоначальном, бытовом

смысле, то оригинал должен перемещаться из одного места в другое. В итоге операции должен оставаться только один экземпляр кода, но находящийся в другом месте, в то время как выполняемая компьютером команда пересылки фактически работает с копией кода. В результате ее выполнения остаются два экземпляра кода — один в источнике, а другой в приемнике. Это дает возможность пересылать один и тот же код из одного и того же источника произвольное количество раз.

Пересылка данных осуществляется командами пересылки `mov`, ввода `in`, вывода `out` и обмена значениями `xchg`, а также командами, осуществляющими манипуляции со стеком.

Команда `mov` используется для пересылки копии кода из команды, регистра или поля памяти в другой регистр или поле памяти. Это аналог оператора присваивания из языков высокого уровня в его простейшем варианте, когда справа находится константа или переменная. В команде могут быть использованы любые регистры общего назначения, индексные, базовые и сегментные регистры процессора. Но на использование в команде сегментных регистров накладываются некоторые ограничения, которые обсуждаются в дальнейшем.

Длина пересылаемого кода определяется по следующим правилам. Если операндом команды является 8-битовый регистр (`al`, `ah`, `bl`, `bh` и т. д.), то пересылается 1 байт; если операнд — это 16-битовый регистр (`ax`, `bx`, и т. д.), то пересылается 2 байта. Если в операции пересылки регистры процессора не участвуют, то во внимание принимается длина поля памяти, которая задается в директиве определения данных. В тех ситуациях, когда и эта возможность определения длины отсутствует, как, например, в рассмотренных ранее (см. 4.2.3) случаях применения косвенной адресации, программист должен включить в команду дополнительный параметр `byte ptr` или `word ptr`. Операнды команды пересылки не должны диктовать противоречивые требования к длине пересылаемого кода. Так, например, в команде не могут одновременно использоваться 8-битовый и 16-битовый регистры.

Если пересылается код, который можно трактовать как отрицательное число, флажок `sf` устанавливается в 1. Если пересылается нулевой код, то в единицу устанавливается флажок `zf`. Флажки `cf`, `af` и `of` этой командой не затрагиваются.

На пересылку данных с помощью команды `mov` накладываются следующие ограничения:

- запрещается прямая пересылка из одного поля памяти в другое;
- запрещается использование в качестве приемника регистра `cs`;
- нельзя загрузить код в сегментный регистр из поля памяти;
- нельзя переслать код из одного сегментного регистра в другой.

Рассмотрим примеры команд пересылки:

```
1. mov   ah,15h          ;пересылается 1 байт, ah:=21₁₀
2. mov   ax,bx           ;пересылается 2 байта, ax:=bx
3. mov   di,cs           ;пересылается 2 байта, di:=cs
4. mov   ah,bx           ;ошибка в команде
```

```
 5. mov   ah,p1          ;пересылается 1 байт, ah:=p1
 6. mov   ax,p1          ;пересылается 2 байта, ax:=p1
 7. mov   p1,ah          ;пересылается 1 байт, p1:=ah
 8. mov   p1,p2          ;ошибка в команде
 9. mov   ds,p1          ;ошибка в команде
10 .mov   ds,es          ;ошибка в команде
11. mov   ax,offset p1   ;пересылка смещения поля p1
12. mov   ax,seg p1      ;пересылка адреса сегмента поля p1
```

Вспоминая, что в Ассемблере действует правило, по которому приемник результата указывается в списке операндов команды на первом месте, получим приведенные в комментариях результаты выполнения этих команд. По первой команде непосредственный однобайтовый операнд 15_{16} копируется в регистр `ah`. По второй команде два байта из регистра `bx` копируются в регистр `ax`. По третьей команде двухбайтовый код из регистра `cs` копируется в регистр `di`. Обратите внимание на то, что похожая команда `mov cs, di` ошибочна, так как регистр `cs` не может быть приемником кода в командах пересылки. Это запрещение объясняется тем, что содержимое регистра `cs` участвует в выборке следующей команды выполняющейся программы. Поэтому произвольная замена его содержимого приведет к непредсказуемой выборке кода, который, очевидно, не является следующей командой программы.

Комментарий к четвертой команде `mov ah, bx` сообщает, что эта команда ошибочна. Ошибка состоит в том, что один операнд `ah` требует пересылки однобайтового кода, а другой операнд `bx` — двухбайтового. Такое противоречивое требование к длине пересылаемого кода со стороны операндов команды *недопустимо*. В следующих двух примерах код выбирается из поля `p1`, которое по определяющей его директиве имеет длину 2 байта. В пятом примере код пересылается в регистр `ah`, поэтому из поля выбирается 1 байт, а в шестом примере он пересылается в регистр `ax`, поэтому из поля выбирается 2 байта. В данных случаях операнды команды не вызывают противоречия при определении длины операнда, как это имеет место в четвертом примере.

Восьмой пример `mov p1, p2` также содержит ошибку. Оба операнда команды определяют поле оперативной памяти. Это невозможно потому, что команда `mov` преобразуется либо в двухадресную команду, либо в команду с непосредственным операндом. Следовательно, один из операндов должен либо быть регистром процессора, либо непосредственным операндом. Если такую пересылку все-таки необходимо осуществить, ее можно организовать с помощью промежуточной записи кода в один из регистров процессора, — например, так: `mov ax, p2`, а затем `mov p1, ax`.

В девятом и десятом примерах приемником является сегментный регистр, на запись в который наложены приведенные ранее ограничения. В команде `mov ds, p1` сделана попытка передать код из поля памяти в сегментный регистр, что запрещено правилами ассемблера, поскольку отсутствует реализующая такие действия машинная команда. Поэтому такую пересылку нужно выполнять через промежуточный регистр: `mov ax, p1`, а затем `mov ds, ax`. Аналогичным образом

запрещена непосредственная пересылка из одного сегментного регистра в другой. Рецепт тот же: использовать промежуточную запись, например, в один из регистров общего назначения `mov ax, es`, а затем `mov ds, ax`.

В последних двух примерах пересылается не код, находящийся в поле `p1`, а компоненты адреса этого поля (сравните с примером 6). В одиннадцатом примере пересылке подлежит внутрисегментное смещение, о чем говорит стоящее перед меткой `p1` ключевое слово `offset`, а в последнем примере пересылается адрес сегмента, так как перед меткой находится ключевое слово `seg`.

Команды ввода и вывода `in` и `out` кратко рассмотрены в 4.2.7. Поскольку для реализации ввода и вывода на практике используются механизмы прерываний, с помощью которых осуществляется обращение к включенным в операционную систему средствам обмена, рассматривать более подробно эти команды не имеет смысла.

Команда `xchg` осуществляет обмен значениями, которые адресуются операндами команды. Результатом выполнения этой команды является не единственный код, а пара пересылаемых при обмене кодов. Кроме того, изменение значения происходит не только у первого операнда, но и у второго. Обмен значениями может происходить между двумя регистрами или между регистром и полем памяти. Рассмотрим пример. Пусть `[ax]` = 00f016, `[di]` = f00016 и выполняется команда `xchg ax, di`. В результате ее выполнения регистры будут содержать следующие коды: `[ax]` = f00016, `[di]` = 00f016. Заметим, что стандартное выполнение такого обмена в Паскале требует трех присваиваний и одной вспомогательной переменной.

4.3.6. Работа со стеком

Стеком называется динамическая структура данных, организованная на базе связного списка, у которого для включения и исключения новых элементов доступен только один конец. Стек можно представлять себе как некий ящик без крышки (рис. 4.28), в который могут помещаться только *однотипные объекты*, например двухбайтовые коды. Первый объект, помещенный в стек, перекрывается вторым, второй перекрывается третьим и т. д. Таким образом, выбрать из стека можно только самый последний поступивший в него объект. Например, на рис. 4.28 первым в стек попал код F0 26_{16}, затем в стек поступил код 1A $F0_{16}$. Этот код перекрыл первоначально попавший в стек. Если теперь потребуется выбрать код из стека, то сначала будет выбран код 1A $F0_{16}$, а только затем код F0 26_{16}. Легко заметить, что объекты выбираются из стека в порядке, противоположном порядку их занесения в стек. В связи с этим говорят, что стек обслуживается по принципу LIFO (от Last In First Out — последний включенный исключается первым). Доступный конец стека, через который объект поступает в стек или выбирается из него, принято называть **вершиной стека**. Основными операциями над стеком являются создание пустого стека, проверка стека на пустоту (то есть имеется ли в стеке хотя бы один объект или нет), а также включение в вершину стека нового объекта и выборка объекта из стека.

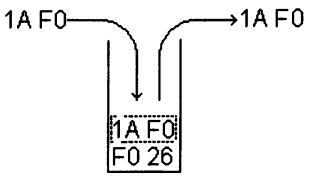


Рис. 4.28. Принцип действия стека

В программировании стек широко используется для временного хранения кодов данных, а также во время обращения к подпрограммам для передачи им значений фактических параметров и обеспечения корректного возврата из подпрограмм. В предыдущем разделе мы видели, что стек используется и для организации прерываний. В связи с такой важной ролью стека в архитектуре процессоров Intel предусмотрен набор программно-аппаратных средств, обеспечивающих все необходимые действия со стеком.

Для реализации стека программист может использовать сегмент стека оперативной памяти и соответствующий программный сегмент. Как уже упоминалось, адрес сегмента стека выбирается из регистра `ss`, а внутрисегментное смещение вершины стека — из регистра `sp`. Таким образом, пара `ss:sp` содержит полный указатель на вершину стека, то есть на код, занесенный в стек последним (рис. 4.29). Элементами программного стека являются машинные слова, то есть двухбайтовые коды.

Если стек пуст, то вершина стека совпадает с его дном. При этом регистр `sp` содержит код $FFEE_{16}$, который автоматически заносится в него перед запуском программы (см. рис. 4.20). Поскольку сегмент стека, как и любой другой сегмент оперативной памяти, имеет размер 64 Кбайт, на первый взгляд кажется, что внутрисегментное смещение вершины пустого стека должно совпадать со смещением для последнего слова сегмента $FFFE_{16}$, а оно равно $FFEE_{16}$. Наблюдаемое различие вызвано тем, что операционная система резервирует в сегменте стеке 16 последних байтов для своих нужд.

При записи в стек списка формальных параметров подпрограммы смещение начального байта этого списка сохраняется в регистре `bp`. Таким образом, пара регистров `ss:bp` содержит указатель на начало списка фактических параметров. Область стека между указателями `ss:bp` и `ss:sp`, которая содержит весь список параметров, называют **кадром стека**. В связи с этим регистр `bp`, содержащий адрес, от которого начинается кадр, называют **указателем базы стека** или **указателем кадра стека** (рис. 4.29). Смещение из этого регистра можно использовать для доступа к отдельным параметрам подпрограммы.

Для работы со стеком в Ассемблере предусмотрено несколько команд. В частности, для копирования слова из регистра или поля памяти в стек применяется команда `push`, а для удаления слова из стека и последующего его размещения в регистре процессора или в поле памяти — команда `pop`. Обе команды имеют один операнд, который может быть регистром процессора или полем оперативной памяти. В команде `push` операнд указывает адрес кода, который должен быть записан в стек, а в команде `pop` — адрес приемника кода, выбранного из его вершины.

Рис. 4.29. Схема сегмента стека

Важной особенностью выполнения работающих со стеком команд является уменьшение содержимого регистра `sp` при включении кода в стек. Это означает, что вершина стека растет в сторону уменьшения адресов оперативной памяти. И наоборот, при выборке из стека содержимое `sp` увеличивается. Рассмотрим эту особенность на конкретном примере. Пусть до выполнения команды включения в стек регистры содержат следующие значения: [ss] = $00A8_{16}$, [sp] = $001C_{16}$ и [ax] = $00FF_{16}$. Пусть далее в сегменте стека по адресу `ss:sp` = 00A8:001C, то есть в вершине стека, находится код $0BAF_{16}$ (рис. 4.30, *а*). Выполнение команды `push ax` происходит следующим образом:

1) осуществляется декремент `sp` на две единицы: [sp] = 001A;
2) производится запись двухбайтового кода 00 FF_{16}, скопированного из регистра `ax`, в новую вершину стека по адресу `ss:sp` = 00A8:001A.

Результат выполнения команды представлен на рис. 4.30, *б*. В поле памяти с адресом $00A9A_{16}$ находится код FF 00_{16} (не забывайте об обратном порядке записи кодов в оперативную память!).

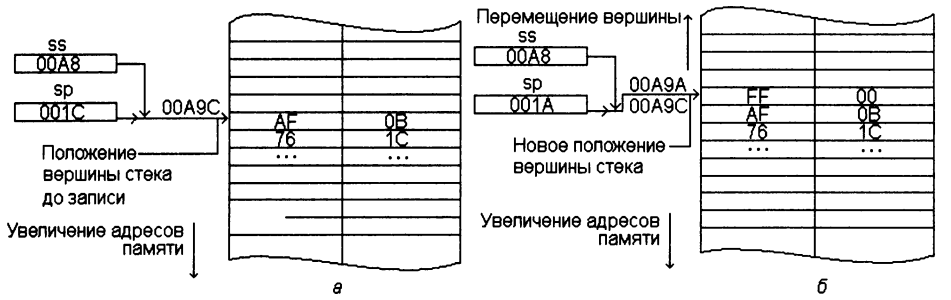


Рис. 4.30. Схема выполнения команды записи в стек push

Выполнение команды `pop ax` в тех же начальных условиях происходит следующим образом (рис. 4.31):

1) осуществляется пересылка кода 0B AF_{16}, выбранного из вершины стека, в указанный в команде регистр `ax`;
2) производится инкремент `sp` на две единицы: [sp] = 001E.

Исходное состояние стека, до выполнения команды `pop ax`, приведено на рис. 4.31, *а*, а результат ее выполнения — на рис. 4.31, *б*.

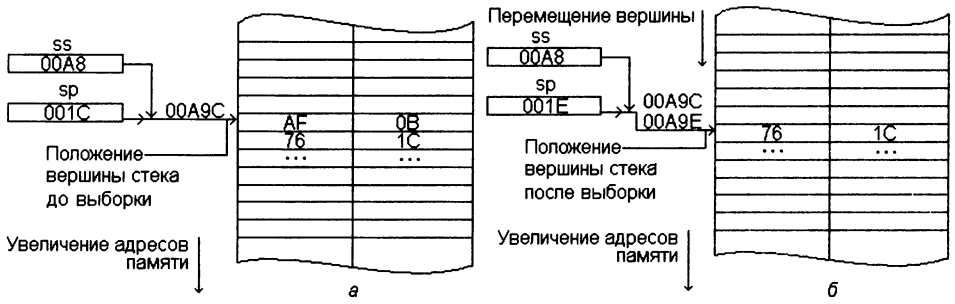


Рис. 4.31. Схема выполнения команды выборки из стека pop

Заметим, что процессор i8086 имеет ограниченное количество регистров, а объем стека во много раз больше. Поэтому вместо предложенной при разборе ошибочных пересылок временной записи в регистр `ax` или какой-либо другой регистр процессора можно организовать временное занесение в стек. Например, вместо последовательности команд `mov ax, p1; mov ds, ax` можно применить эквивалентные по результатам команды `push p1` и `pop ds`. Но при выборе способа организации пересылки не следует забывать об эффективности — использование стека для промежуточного хранения связано с обращением к оперативной памяти и, следовательно, требует больше времени, чем промежуточная запись в регистр процессора.

На рис. 4.27, *б* показан использующий команду `push` фрагмент стандартного начального участка программы на Ассемблере, который обеспечивает правильное выполнение программы и корректный возврат в операционную систему. Отметим, что эту последовательность действий иногда называют **инициализацией программы**. По принятым в операционной системе правилам требуется, чтобы любая программа в самом начале своего выполнения:

- сохранила в стеке содержимое сегментного регистра `ds` (команда `push ds`);
- в следующее слово стека занесла нулевой код (команды `sub ax, ax` и `push ax`);
- занесла в регистр `ds` адрес сегмента данных программы, соответствующий метке этого сегмента (команды `mov ax, dtseg` и `mov ds, ax`, здесь предполагается, что `dtseg` — метка директив, ограничивающих сегмент данных).

Команды `push` и `pop` оперируют с двухбайтовыми кодами. Две другие стековые команды, `pusha` и `popa`, работают с содержимым сразу восьми регистров процессора, `ax`, `cx`, `dx`, `bx`, `sp`, `bp`, `si` и `di`, записывая их в стек в приведенном порядке,

а выбирая в обратном. Команду `pusha` используют при вызове подпрограммы для сохранения текущего состояния регистров вызывающей программы или подпрограммы. После возврата из подпрограммы использование единственной команды `popa` позволяет восстановить обстановку, существовавшую на момент обращения к подпрограмме. Для полноты картины можно сохранить еще и состояние регистра флажков. Для этого используются команды `pushf` и `popf`. Все команды (`pusha`, `popa`, `pushf` и `popf`) являются безадресными.

4.3.7. Сложение и вычитание

Действия сложения и вычитания могут выполняться над 8- и 16-битовыми беззнаковыми и знаковыми целочисленными данными, которые могут находиться в регистрах процессора или в полях оперативной памяти. В некоторых случаях могут использоваться и непосредственные операнды. Команды процессора, связанные со сложением и вычитанием, приведены в табл. 4.20. В этой таблице `p` означает операнд — приемник результата, а `i` — второй участвующий в действии операнд.

Таблица 4.20. Команды сложения и вычитания процессора i8086

Группа	Название	Команда	Действие
Команды сложения	Сложение	`add p, i`	`p := p + i`
	Сложение с переносом	`adc p, i`	`p := p + i + cf`
	Инкремент	`inc p`	`p := p + 1`
Команды вычитания	Вычитание	`sub p, i`	`p := p - i`
	Вычитание с займом	`sbb p, i`	`p := p - i - cf`
	Декремент	`dec p`	`p := p - 1`
	Изменение знака	`neg p`	`p := -p`
	Сравнение	`cmp p, i`	`p - i`

С командами сложения, вычитания, инкремента и декремента мы уже сталкивались при обсуждении форматов машинных команд. Необходимость в командах сложения с переносом и вычитания с займом, а также особенности их выполнения рассматриваются немного позже. А пока отметим, что при сложении с переносом к результату добавляется бит флажка `cf`, а при вычитании с займом результат дополнительно уменьшается на этот же бит. По команде `neg p`, как это следует из таблицы, у единственного ее операнда изменяется знак. Команду `neg` удобно использовать, например, для получения модуля числа. В последней из приведенных в табл. 4.20 команде сравнения `cmp p, i` результат вычитания *не записывается* в приемник и, следовательно, теряется. Фактически вычитание осуществляется только с целью получения соответствующих значений флажков состояния, по которым можно определить дальнейшие действия в программе.

Все приведенные в таблице команды формируют соответствующие полученному результату значения флажков `sf` и `zf`, с помощью которых организуются нужная

логика обработки данных, необходимые ветвления и циклы. Формируются также значения остальных флажков состояния `cf`, `of`, `af` и `pf`, с помощью которых организуется контроль за допустимостью полученных результатов в их знаковой, беззнаковой или какой-либо другой трактовке (см. 4.1.6 и табл. 4.2).

4.3.8. Умножение и деление

В процессоре i8086 предусмотрены по два варианта команд деления и умножения. Это команды `mul` (от multiply — умножение) и `div` (от division — деление) для беззнаковых операндов, а также команды `imul` (от integer multiply) и `idiv` (от integer division) — для знаковых. Операнды команд умножения и деления могут быть как однобайтовыми так и двухбайтовыми.

Вначале рассмотрим важную особенность умножения, которая оказывает существенное влияние на машинную организацию обсуждаемых действий. Эта особенность проиллюстрирована примерами (рис. 4.32). Вначале вспомним, что при беззнаковой трактовке в однобайтовом поле могут находиться коды чисел от 0 до 255_{10}, а в двухбайтовом — от 0 до 65535_{10}. На рис. 4.32, *а* приведен пример умножения десятичных (*слева*) чисел и соответствующих им двоичных (*справа*) кодов, когда результат, так же как и сомножители, может быть записан в один байт. На рис. 4.32, *б* приведен пример, когда сомножители еще не очень велики (38_{10} и 19_{10}), но результат (722_{10}) уже значительно превосходит максимально возможное для одного байта значение 255_{10}. Справа представлено это же действие в двоичных кодах. Видно, что результат в один байт не помещается. В рамку взяты две цифры результата, для которых не хватает места в байте. По-видимому, такая ситуация при умножении возникает довольно часто. В общем случае можно утверждать, что при умножении двузначных десятичных чисел результат оказывается трех- или четырехзначным. Аналогичным образом при умножении двух 8-битовых двоичных кодов результат может занимать до 16 битов памяти.

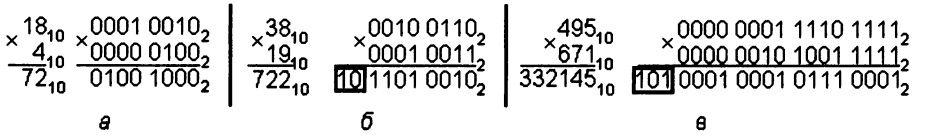


Рис. 4.32. Особенность выполнения операции умножения

На рис. 4.32, *в* показана такая же особенность, возникающая при умножении чисел, больших, чем 255_{10}, то есть при умножении чисел, коды которых размещаются не менее чем в двухбайтовых полях. Как видно из приведенного на рисунке примера, результат умножения в два байта может не поместиться. Очевидно, что при умножении двух 16-битовых чисел для записи результата может потребоваться до 32 битов.

С учетом этого обстоятельства умножение в процессоре i8086 организовано следующим образом. Если умножаются 8-битовые сомножители, то один из них до начала выполнения умножения должен быть записан в регистр `al`. Адрес второго

сомножителя I8 указывается в команде умножения: mul I8. Результат всегда записывается в регистр ax. Схему выполнения такого умножения можно кратко записать в условном, паскалеподобном виде: ax := [al] × I8. В развернутом виде схема приведена слева вверху на рис. 4.33. В команде, условной формуле и на рисунке I8 означает заданный в команде 8-битовый сомножитель или его адрес.

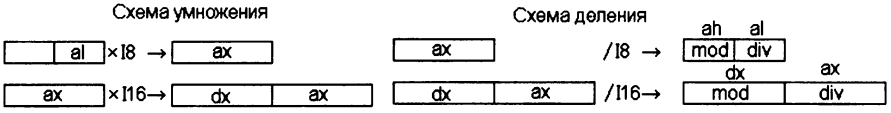


Рис. 4.33. Схемы выполнения операций умножения и деления

Очевидно, что для деления должна иметь место обратная ситуация. Делимое, которое, в частности, может быть результатом предшествующего умножения, должно быть до выполнения деления записано в 16-битовый регистр ax. 8-битовый делитель указывается в команде деления div I8. Частное и остаток целочисленного деления, полученные в результате выполнения команды, всегда заносятся в регистр ax. При этом частное занимает регистр al, а остаток от деления — регистр ah. Схема выполнения деления, которую можно условно записать в виде al := [ax] div I8; ah := [ax] mod I8, показана на этом же рис. 4.33, *справа вверху*. В краткой записи и на рисунке div и mod — это принятые в Паскале обозначения операции вычисления частного и остатка от целочисленного деления.

Рассмотрим примеры команд, задающих 8-битовое умножение и деление:

```
mov   al,4     ;пересылка в al первого сомножителя 4
mov   cl,27    ;пересылка в cl второго сомножителя 27
mul   cl       ;произведение 4 · 27 заносится в регистр ax
mov   bl,5     ;пересылка в bl делителя 5
div   bl       ;частное 21 в al, остаток 3 в ah
```

В результате выполнения первых двух команд создается возможность выполнить умножение. Первый сомножитель, число 4, записывается в фиксированный регистр al. Второй сомножитель может быть записан в любой 8-битовый регистр или же в байт памяти. В рассматриваемых примерах для записи второго сомножителя — числа 27 — выбран регистр cl. Результат получается с помощью команды умножения, в которой указывается адрес второго сомножителя mul cl. Отметим, что в данном случае значение, находящееся в регистре ah, не играет никакой роли. Поскольку в команде указан 8-битовый операнд, в операции участвует только содержимое al. Результат занимает весь регистр ax. Если в дальнейшем возникнет необходимость выполнить его деление, то делимое уже занимает необходимое положение. Останется задать только 8-битовый делитель. В обсуждаемых примерах осуществляется деление полученного в регистре ax результата на число 5. Для этого командой пересылки mov bl,5 делитель 5 помещается в регистр bl, а затем этот регистр указывается в качестве операнда команды деления: div bl. В результате частное от деления 21 займет регистр al, а остаток 3 — регистр ah.

В связи с тем что регистры процессора i8086 16-битовые, а в результате умножения двух 16-битовых сомножителей произведение может потребовать 32 бита, разработчики процессора решили записывать такое произведение сразу в два регистра, dx и ax. При этом младшие 16 разрядов результата записываются в регистр ax, а старшие 16 разрядов — в регистр dx.

Общая схема 16-битового умножения примерно такая же, что и 8-битового. Один из сомножителей до выполнения команды необходимо записать в регистр ax. Адрес второго сомножителя указывается в команде умножения. 32-битовый результат занимает указанным ранее способом пару регистров dx:ax. Схему такого умножения можно кратко изобразить в условном виде (dx, ax) := [ax] × I16, где I16 означает заданный в команде 16-битовый операнд. Эта же схема в более подробном виде приведена на рис. 4.33, *слева внизу*.

16-битовое деление организуется следующим образом. 32-битовое делимое до начала деления должно находиться в паре регистров dx:ax. Поскольку процессор i8086 относится к 16-битовой архитектуре, 32-битовые коды данных в таком процессоре большинством команд не воспринимаются и не формируются. В команде деления предполагается, что такое делимое получено в результате одной из предшествующих команд умножения или же сформировано с помощью специальных приемов. Адрес 16-битового делителя указывается в команде. Частное от деления занимает регистр ax процессора, а остаток от деления размещается в регистре dx. Условно схему деления можно изобразить в виде ax := (dx, ax) div I16; dx := (dx, ax) mod I16. Полная схема изображена на рис. 4.33, *внизу справа*.

Примеры команд, задающих 16-битовое умножение и деление:

```
mov   ax,356    ;пересылка в ax первого сомножителя 356
mov   di,413    ;пересылка в di второго сомножителя 413
mul   di        ;произведение 356 · 413 заносится в пару (dx,ax)
mov   si,509    ;пересылка в si делителя 509
div   si        ;частное 288 в ax, остаток 436 в dx
```

Важно понимать, что при выполнении умножения результат может занимать всего 1 байт при 8-битовом операнде и 2 байта при 16-битовом операнде. Пример такого умножения приведен на рис. 4.32, *а*. Чтобы выяснить, какая именно ситуация фактически имеет место, после выполнения умножения нужно проанализировать значение флажка cf. Если cf = 1, то результат занимает 16 или же 32 бита, в противном случае он занимает 8 или 16 битов в зависимости от длины операнда команды умножения.

Выполнение команды деления может быть сопряжено с исключительными ситуациями. Дело в том, что делитель может оказаться равным нулю, и, следовательно, выполнить эту операцию процессор не способен в принципе. Как известно, в этом случае вникает связанное с исключительной ситуацией внутреннее прерывание типа 0. Кроме того, делитель может быть настолько малым, а делимое настолько большим, что частное не поместится в выделенное ему место. Пусть, например, при 8-битовом делении делимое равно $16\,000_{10}$. Такое делимое может быть записано в регистр ax, и, следовательно, оно может участвовать в 8-битовом

делении. Пусть далее делитель равен 2_{10}, он также может участвовать в этой операции. А вот результат этого деления $8\,000_{10}$ в 8-битовый регистр `al` записан быть не может. В этом случае также возникает внутреннее прерывание типа 0. В случае равного нулю делителя нужно искать логическую ошибку в программе. А во втором случае, после того как произошло прерывание и проверка делителя показала, что он не равен нулю, ситуацию можно исправить, перейдя к 16-битовому делению.

Аналогичная ситуация может возникнуть и при 16-битовом делении, когда делимое и делитель таковы, что частное в регистр `ax` не помещается. Простого решения этой проблемы в рамках 16-битовой архитектуры не существует. Нужно либо переходить к работе с процессором, обладающим 32-битовым машинным словом, либо привлекать специальные алгоритмы выполнения арифметических операций над так называемыми многоразрядными целыми числами. Один из приемов работы с многоразрядными слагаемыми обсуждается далее, в 4.3.8.

В целом выполнение умножения и деления `imul` и `idiv` для знаковых операндов происходит по тем же самым схемам, что и у беззнаковых. При этом имеется естественное отличие от беззнаковых команд в диапазоне операндов и результатов, соответствующих знаковым 8- и 16-битовым целым числам. Поэтому для проверки принадлежности результата допустимому диапазону используется не флажок `cf`, а флажок `of`.

При выполнении знакового целочисленного деления нужно помнить о том, что эта операция имеет неоднозначный результат. Пусть нужно выполнить целочисленное деление числа x на число y. По определению целочисленного деления, частное z и остаток от деления r подбираются таким образом, чтобы $x = zy + r$. Пусть, например, число $x = -26$ нужно разделить нацело на число $y = +7$. Для этой операции, в принципе, возможны два ответа. В первом варианте частное $z = -4$ и остаток $r = +2$. Проверим: $-4 \cdot (+7) + 2 = -26$. Во втором варианте частное $z = -3$ и остаток $r = -5$. Проверим: $-3 \cdot (+7) - 5 = -26$. В процессорах Intel при выполнении целочисленного знакового деления действует правило, в соответствии с которым *знак остатка всегда выбирается тем же самым, что и знак делимого.* Таким образом, правильным является последний вариант. Об этом нужно помнить, чтобы не получить разночтений при трактовке результатов знакового деления.

Для делимого, которое не было получено в результате предшествующего умножения, необходимо правильно сформировать его код в регистре `ax` или же в паре регистров (`dx`, `ax`). Пусть, например, делимым является число $+35_{10} = 0010\ 0011_2$. Код делимого должен находиться в регистре `ax`. Если просто получить или переслать код $0010\ 0011_2$ в регистр `al`, то неизвестный код, который находится в `ah` и участвует в делении, может привести к искажению конечного результата. Просто записать в регистр `ah` нулевой код также нельзя, поскольку делимое может быть отрицательным, и тогда в `ah` должны находиться единичные биты. Поэтому правильная организация программы требует от программиста внимательного отслеживания расширения знакового разряда.

В табл. 4.21 приведены последовательности команд, которые обеспечивают корректное формирование делимого как в случае подготовки к беззнаковому, так и при подготовке к знаковому делению.

Таблица 4.21. Корректная подготовка к делению

Длина операндов	Беззнаковое деление	Знаковое деление
8 битов	`mov al`, делимое	`mov al`, делимое
	`mov ah, 0`	`cbw`
	`div`, делитель	`idiv`, делитель
16 битов	`mov ax`, делимое	`mov ax`, делимое
	`mov dx, 0`	`cwd`
	`div`, делитель	`idiv`, делитель

В случае беззнакового деления сначала в регистр `al` пересылается делимое. Затем необходимо заполнить `ah` нулевым кодом, что и осуществляется командой `mov ah`. И только после этого записывается команда деления. Точно так же организуется и 16-битовый вариант деления, но в подготовке участвуют регистры `ax` и `dx`.

Подготовка к знаковому делению связана с копированием знакового бита из регистров `al` или `ax` в регистры `ah` и `dx` соответственно. Эту операцию можно автоматизировать, применяя команды `cbw` (от convert byte to word — преобразовать байт в слово) или `cwd` (от convert word to double — преобразовать слово в двойное слово). Вначале делимое записывается в соответствующий его длине регистр `al` или `ax`. Затем по команде `cbw` код из регистра `al` расширяется до регистра `ax` дублированием его знакового бита или по команде `cwd` знаковый бит кода из регистра `ax` заполняет регистр `dx`. Таким образом, в каждом случае формируется нужный код делителя. Далее применяется команда `idiv`, в которой указывается нужный делитель.

4.3.9. Организация линейных программ на машинном уровне

Теперь мы можем приступить к разработке линейных машинных программ, записывая их на Ассемблере.

Обработка 8-битовых данных

По принятым в большинстве языков программирования правилам, которые естественно вытекают из необходимости на машинном уровне (да и в обычной жизни) где-то хранить обрабатываемые данные и результаты, а в командах указывать адреса их хранения, необходимо предварительно выделить поля для всех используемых величин. Для этого в сегменте данных программы с помощью директив определения данных резервируются рабочие поля. Целесообразно в качестве меток рабочих полей выбирать имена переменных, участвующих в вычислениях. Поскольку длина данных считается равной одному байту, для резервирования следует использовать директиву `db`. Так как значения переменных, участвующих в вычислениях, еще не известны, в директиве `db` следует использовать параметр «`?`», резервирующий пустые рабочие поля. Конкретные значения в эти

поля попадут в результате выполнения операций ввода данных уже в процессе выполнения программы. Итак, директива резервирования рабочего поля для хранения значения, например, переменной a должна выглядеть следующим образом: a db ?. Аналогичные директивы должны определять рабочие поля для всех используемых в программе переменных.

Как уже отмечалось, все директивы определения данных целесообразно группировать в отдельном программном сегменте данных. Выберем для ограничивающих этот сегмент директив метку dtsg. Весь программный сегмент данных занимает первые семь строк листинга 4.2. Далее (строки 8–10 листинга) следует программный сегмент стека, структура которого обсуждалась ранее.

Листинг 4.2. Пример линейной программы

```
 1.   dtsg   segment   'data' ;начало сегмента данных
 2.   a      db        ?       ;8-битовые данные
 3.   b      db        ?
 4.   c      db        ?
 5.   d      db        ?
 6.   x      db        ?
 7.   dtsg   ends              ;конец сегмента данных
-------------------------------------------------
 8.   stsg   segment   'stack';начало сегмента стека
 9.   dw     512 dup(?)       ;выделение памяти под стек
10.   stsg   ends             ;конец сегмента стека
-------------------------------------------------
11.   cdsg   segment   'code' ;начало сегмента кода
12.          assume    cs:cdsg,ds:dtsg,ss:stsg,es:nothing
13.          push      ds      ;стандартное начало программы
14.          sub       ax,ax   ;стандартное начало программы
15.          push      ax      ;стандартное начало программы
16.          push      dtsg    ;стандартное начало программы
17.          pop       ds      ;стандартное начало программы
18.          ...       ;организация ввода исходных данных
19.          mov       al,2    ;пересылка в al сомножителя 2
20.          mul       a       ;умножение 2 на a,[ax]=2a
21.          mov       cx,ax   ;пересылка 2a в cx,[cx]=2a
22.          mov       al,c    ;пересылка в al сомножителя c
23.          mul       b       ;умножение b на c,[ax]=bc
24.          add       ax,cx   ;сложение произведений,[ax]=2a+bc
25.          mov       bl,d    ;пересылка в bl уменьшаемого
26.          sub       bl,3    ;получение знаменателя,[bl]=d-3
27.          div       bl      ;частное в al, остаток в ah
28.          mov       x,al    ;пересылка результата в поле x
29.          ...               ;организация вывода результата
30.          cdsg    ends      ;конец сегмента кода
```

Описывающие требуемые вычисления команды программы принято группировать в программном сегменте кода. Для ограничивающих этот сегмент директив

выберем метку cdsg. Обычно директиву assume размещают в начале сегмента кода (строка 13), хотя это и не обязательно.

Следующие пять строк этого сегмента занимают обсуждавшиеся выше начальные действия по инициализации программы, которые диктуются правилами операционной системы (см. 4.3.6 и рис. 4.27, *б*). Отметим только, что в них применен основанный на командах push и pop альтернативный вариант формирования в регистре ds адреса программного сегмента данных (строки 16 и 17 листинга).

Далее в программе следует организовать ввод значений переменных *a*, *b*, *c* и *d*, для которых должен быть выполнен расчет значения x. Как попадают исходные данные в рабочие поля, мы обсуждать не будем, потому что вопросы организации обмена с внешними устройствами довольно сложно решаются даже на языках высокого уровня. Ранее упоминалось, что в Ассемблере эти проблемы решают с помощью обращения к стандартным средствам обмена, предусмотренным в операционной системе. В листинге 4.2 программы организация ввода заменена многоточием (строка 18).

Общая схема вычислений в данном случае проста и диктуется только правилами старшинства операций. Искомое значение x представлено дробью. Для вычисления ее значения вначале следует организовать вычисление числителя, а затем знаменателя (можно и наоборот). В числителе сначала вычисляются два произведения, $2a$ и bc, а затем находится их сумма. Знаменатель находится в одно действие вычитания. После чего целочисленным делением находится искомое значение x, которое следует переписать в подготовленное для него поле памяти.

Начнем с вычисления слагаемого $2a$. Его можно найти простым суммированием $a + a$. Однако чтобы продемонстрировать стандартно используемые приемы, получим это значение с помощью умножения. По правилам выполнения умножения в системе команд процессора i8086 один из сомножителей должен находиться в регистре al (для 8-битовых сомножителей). Поэтому перешлем первый сомножитель 2 в регистр al. А в следующей команде умножения укажем второй сомножитель *b*. Результат $2a$ получается в регистре ax. Таким образом найдено первое слагаемое (строки 19 и 20 листинга).

Второе слагаемое можно найти точно таким же способом. Но проблема в том, что, получая второе слагаемое *bc* в регистре ax, мы тем самым уничтожим уже находящееся там первое слагаемое $2a$. Следовательно, его нужно *предварительно* переписать для промежуточного хранения в любой свободный 16-битовый регистр. Выберем для этого регистр cx (строка 21 листинга).

Строки 22 и 23 листинга содержат уже обсуждавшиеся команды, с помощью которых в регистре ax формируется второе слагаемое числителя *bc*.

На данный момент рассуждений имеется следующая ситуация. В регистре ax находится первое слагаемое числителя, а в регистре cx — его второе слагаемое. Результат можно получить и в регистре cx (командой add cx, ax) и в регистре ax (командой add ax, cx). Возникает вопрос: какой вариант выбрать? Если проанализировать те действия, которые придется выполнять в дальнейшем, становится ясно, что целесообразно организовать получение числителя в регистре

ax, так как потребуется выполнить его деление на знаменатель. Поэтому для получения числителя выбрана команда add ax, cx (строка 24 листинга).

Теперь займемся знаменателем. Уменьшаемое *d* находится в поле памяти. Из него требуется вычесть число 3. Это можно сделать командой вычитания с непосредственным операндом sub d, 3. Но в этом случае записываемый в поле d результат вычитания уничтожит значение уменьшаемого. В программировании действует негласное правило, в соответствии с которым полученные путем ввода значения исходных данных программы изменять категорически не рекомендуется. Поэтому уменьшаемое *d* следует предварительно переписать в другое место, например, в регистр bl, как это сделано в строке 25 листинга. Полученный с помощью команды вычитания (строка 26) в регистре bl знаменатель дроби теперь можно указать в качестве операнда команды деления (строка 27). Результаты деления формируются в регистре ax. Далее интересующее нас частное *x* следует переслать из регистра al в приготовленное для него поле в сегменте данных (строка 28). Организация вывода полученных результатов в программе также опущена.

В связи с предшествующим обсуждением команд умножения и деления возникают некоторые вопросы. В частности, нужно ли в данном случае применять специальные меры по формированию делимого, как это показано в табл. 4.22? Следует вспомнить, что такие действия организуются только в том случае, когда делимое получается не как результат предшествующего умножения. Поскольку в данном случае числитель получен именно с помощью предшествующих умножений, ответ на данный вопрос очевиден.

А вот вопрос: «Какие проблемы могут возникнуть при выполнении программы?» — необходимо задавать себе в любом случае анализа программы, а особенно если в ней имеются операции деления. Возможно ли возникновение ситуации, в которой знаменатель окажется равным нулю? Очевидно, возможно. Достаточно в качестве значения переменной *d* задать число 3. Кроме того, возможен вариант, когда результат деления не поместится в регистр al. В связи с этим, строго говоря, предложенный вариант программы не является корректным. Перед выполнением деления необходимо организовать проверку на равенство знаменателя нулю. Далее нужно каким-то образом проверить, поместится ли результата деления в регистре al.

Поскольку организация любой проверки переводит алгоритм из класса линейных в класс алгоритмов с ветвлением, сейчас мы ограничимся выводом о ненадежности полученного варианта программы, а вопросами организации ветвлений займемся позднее.

Действия с многоразрядными данными

Более сложной оказывается разработка программы, если известно, что участвующие в вычислениях данные относятся к 16-битовым. Дело в том, что после умножения 16-битовых чисел *b* и *c* результат оказывается 32-битовым. С этим 32-битовым числом нужно сложить другое число 2*a*, также, возможно, 32-битовое. А теперь мы должны вспомнить, что в системе команд 16-битового процессора i8086 команда сложения может оперировать только с 8- или 16-битовыми операндами.

Данные, длина которых превышает длину машинного слова процессора, принято называть **многоразрядными**. Обработка таких данных стандартными командами процессора невозможна. Поэтому приходится прибегать к специальным методам и алгоритмам, обеспечивающим корректность их обработки. Чтобы понять, в чем здесь проблема, рассмотрим конкретный пример, изображенный на рис. 4.34. Пусть требуется сложить 32-битовый код, занимающий пару регистров (dx, ax), и такой же код в паре (bx, cx). Пример конкретных значений кодов, находящихся в этих регистрах, показан на рис. 4.34, *слева*, а соответствующие им десятичные числа – *справа*. Для получения результата попробуем по отдельности сложить младшие разряды слагаемых, находящиеся в регистрах ax и cx, и старшие разряды из регистров dx и bx. Для такого сложения можно использовать стандартную команду сложения add. Получаем последовательность команд, которая как будто решает поставленную задачу:

```
add ax,cx    ;ax:=[ax]+[cx]
add dx,bx    ;dx:=[dx]+[bx]
```

Результат задаваемого этими командами раздельного сложения приведен в третьей строке рис. 4.34.

Рис. 4.34. Многоразрядное сложение с применением команды adc

Однако если мы переведем полученный двоичный код в десятичную систему (292 912 974_{10}) и сравним его с результатом непосредственного сложения (292 978 510_{10}), то окажется, что результаты не совпадают. Причина этого несовпадения очевидна. При сложении находящихся в регистрах ax и cx кодов возникает единица переноса из их старшего разряда. На рис. 4.34 биты, дающие эту единицу переноса, подчеркнуты. При раздельном, несвязном сложении младших и старших разрядов 32-битовых кодов эта единица переноса оказывается утерянной.

В системе команд процессора i8086 имеется специальная команда сложения, с помощью которой можно учитывать возникновение переносов между занимающими 16-битовые регистры группами разрядов многоразрядных чисел. Это упоминавшаяся ранее команда adc сложения с переносом. Ее выполнение, приведенное в табл. 4.22, отличается от выполнения команды add тем, что к стандартным

образом полученной сумме добавляется значение флажка cf, всегда совпадающее с битом переноса из старшего разряда результата.

С учетом сказанного приходим к следующему решению задачи. Сначала, как и в первом варианте, складываются младшие разряды кодов из регистров ax и cx. Если бит переноса возникает, то cf = 1, а если его нет, то cf = 0. Таким образом, добавление значения флажка cf при последующем сложении старших разрядов с помощью команды adc в любом случае оказывается эквивалентным учету бита переноса. Итак, поставленную задачу решает такая последовательность действий:

```
add ax,cx     ;ax:=[ax]+[cx]
adc dx,bx     ;dx:=[dx]+[bx]+бит переноса
```

Результат сложения формируется в паре регистров (dx, ax). В нижней строке рис. 4.34 приведен результат правильного сложения кодов. В третьей и четвертой строках рисунка в рамку включены участки кодов, затронутые влиянием обсуждаемой единицы переноса. В третьей строке эта единица не учтена, соответственно, получен неправильный результат. А в четвертой строке она учтена с помощью команды adc. Как и следовало ожидать, разница между правильным и неправильным результатами оказалась равной $65\,536_{10}$ Именно такое значение в десятичной системе счисления имеет утерянная единица переноса из 15-го в 16-й разряд: $1 \cdot 2^{16} = 65536_{10}$.

Если требуется выполнить вычитание для 32-битовых кодов, то его, очевидно, можно выполнить по той же самой схеме, но при этом флажок cf, отражающий возможный факт займа во время вычитания, учитывается командой sbb, которая также приведена в табл. 4.22.

Описанный способ выполнения сложения и вычитания для 32-битовых кодов можно применять и для чисел произвольной разрядности, коды которых занимают последовательности машинных слов в оперативной памяти. При этом младшие слова кодов складываются командой сложения add, а каждая последующая пара слов — командой adc, учитывающей возможный перенос от предыдущей пары. Точно так же производится и вычитание многоразрядных кодов. Вычитание для младших слов выполняется командой sub, а вычитание для каждой следующей пары слов — командой sbb. Более сложно, но примерно по тому же самому принципу организуется умножение и деление для многоразрядных чисел.

Обработка 16-битовых данных

Теперь мы можем рассмотреть приведенный в листинге 4.3 более сложный вариант решения поставленной задачи, в котором значения переменных занимают двухбайтовые поля.

Первое отличие от рассмотренного ранее варианта в том, что резервирование памяти выполняется директивой dw, которая выделяет под рабочее поле 2 байта памяти. Все остальные отличия связаны с использованием в операциях умножения и деления 16-битового операнда, а не 8-битового. В частности, для организации промежуточного хранения 32-битового произведения 2a нужны два 16-битовых

регистра bx и cx и две команды пересылки (строки 22 и 23 листинга 4.3), а для получения числителя приходится выполнять описанное в предыдущем разделе многоразрядное сложение (строки 25 и 26 листинга 4.3).

Листинг 4.3

```
 1.   dtsg   segment   'data'       ;начало сегмента данных
 2.   a      dw        ?            ;16-битовые данные
 3.   b      dw        ?
 4.   c      dw        ?
 5.   d      dw        ?
 6.   x      dw        ?
 7.   dtsg   ends                   ;конец сегмента данных
 ------------------------------------------------
 8.   stsg   segment   'stack'      ;начало сегмента стека
 9.          dw        512 dup(?)   ;выделение памяти под стек
10.   stsg   ends                   ;конец сегмента стека
------------------------------------------------
11.   cdsg   segment   'code'       ;начало сегмента кода
12.          assume    cs:cdsg,ds:dtsg,ss:stsg,es:nothing
13.          push      ds           ;стандартное начало программы,
14.          sub       ax,ax        ;
15.          push      ax           ;
16.          push      dtsg         ;
17.          pop       ds           ;
18.          ...       ;организация ввода исходных данных
19.          mov       ax,2;        ;пересылка в ax сомножителя 2
20.          mul       a            ;произведение 2a в паре [dx,ax]
21.          mov       cx,ax        ;пересылка младших разрядов 2a
22.          mov       bx,dx        ;2a в паре [bx,cx]
23.          mov       ax,c         ;пересылка в ax сомножителя c
24.          mul       b            ;произведение bc в паре [dx,ax]
25.          add       ax,cx        ;сложение младших разрядов
26.          adc       dx,bx        ;числитель в паре [dx,ax]
27.          mov       bx,d         ;подготовка уменьшаемого в bx
28.          sub       bx,3         ;знаменатель b-3 в bx
29.          div       bx           ;частное в ax, остаток в dx
30.          mov       x,ax         ;пересылка результата в поле x
31.          ...                    ;организация вывода
32.          cdsg      ends
```

4.3.10. Команды передачи управления

Как мы выяснили ранее, в линейных программах нужный порядок выполнения команд обеспечивается за счет автоматического увеличения содержимого регистра ip на длину выполняемой команды. Организация ветвлений основана на пропуске групп команд, связанных с не выбранными ветвями, и переходом к первой команде выбранной ветви. Организация цикла связана с возвратом

к команде, начинающей цикл, которая ранее уже хотя бы один раз была выполнена. И в том, и в другом случае должен произойти переход к команде, которая не следует непосредственно за текущей. Таким образом, для организации ветвлений и циклов необходимо принудительно изменять линейный порядок выполнения команд программы. Для этого в системе команд процессора i8086 предусмотрены команды, которые называются **командами перехода**. Применяется также другое название: **команды передачи управления**, так как можно считать, что управление процессором передается на другой участок программы или в общем случае — другой программе.

ВНИМАНИЕ

Команды передачи управления (перехода) принудительно загружают в регистры cs и ip новые значения, тем самым обеспечивается выборка не следующей по порядку команды программы, а команды, адрес которой оказался записанным в регистры cs:ip.

Для реализации такой замены все команды перехода в машинном формате содержат в качестве операнда указатель команды, которая должна быть выполнена следующей (к которой необходимо выполнить переход, которой нужно передать управление). Так, например, система команд процессора i8086 содержит команду с кодом операции EA_{16} и мнемокодом `jmp` (от jump — прыжок). По этой команде выполняется переход к команде, заданной ее непосредственным четырехбайтовым операндом. Общая длина команды составляет 5 байтов. Рассмотрим, например, команду EA 00 01 00 $A0_{16}$, ее операндом является код 00 01 00 $A0_{16}$. Первые два байта операнда 00 01_{16} являются новым содержимым регистра `ip`, а следующие два 00 $A0_{16}$ — новым содержимым регистра `cs`. Используемый порядок записи указателя в команде является следствием принципа обратного порядка записи кодов в оперативную память: старшие байты поля операнда содержат более важную информацию — код из регистра `cs`. В результате выполнения команды в регистры `cs` и `ip` загружаются новые значения: [cs] = $A000_{16}$ и [ip] = 0100_{16}. Еще раз обратите внимание на обратный порядок записи кодов и его роль теперь уже в формировании содержимого регистров `cs` и `ip`.

В ассемблерном формате кроме мнемокода команды переходов содержат либо указатель (`jmp A000:0100`), либо метку (`jmp M1`) той команды, к которой нужно выполнить переход.

Пример выполнения команды перехода показан на рис. 4.35. Пусть в некоторый момент времени регистры `cs` и `ip` содержат указатель 00A8:001C (*слева*), соответствующий физическому адресу ($00A9C_{16}$). По этому адресу расположена обсуждавшаяся ранее команда с кодом EA 00 01 00 $A0_{16}$. Во время ее выполнения производится загрузка нового содержимого регистров `cs` и `ip` (*справа*), а когда процессор по стандартному алгоритму начинает выполнять следующую команду, он автоматически обращается не по адресу 00A8:0022 ($00AA2_{16}$), в котором находится команда, следующая за командой перехода, а по адресу, который задан указателем из команды A000:0100 ($A0100_{16}$).

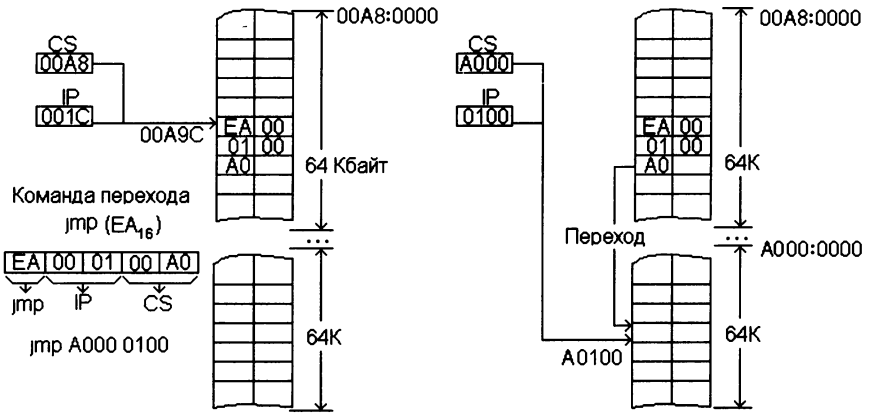


Рис. 4.35. Выполнение межсегментного перехода

Классификация переходов

Переходы бывают **межсегментными** и **внутрисегментными**. Внутрисегментные переходы, в свою очередь, делятся на **близкие** и **короткие**. Для организации межсегментных переходов заменяется и содержимое регистра `ip`, и содержимое регистра `cs`. Именно вследствие замены содержимого `cs` и происходит переход из сегмента, в котором находится команда перехода, в другой сегмент, начинающийся в другом месте, с другого адреса. Отсюда и название таких переходов — межсегментные, или **дальние** переходы. На рис. 4.35 приведен пример межсегментного перехода.

Во время выполнения внутрисегментных переходов содержимое регистра `cs` не изменяется, меняется только содержимое регистра `ip`. Адрес такого перехода занимает в команде один или два байта. Если он занимает два байта, то переход считается близким. Близкий переход осуществляется в пределах сегмента, содержащего команду перехода, то есть в пределах 64 Кбайт. Если адрес перехода занимает в команде только один байт, то переход считается коротким. Короткий переход возможен только на расстояния, не превышающие 255 байтов. В соответствии с выполняемым переходом указатели (адреса переходов) бывают межсегментные (дальние) и внутрисегментные — близкие или короткие. Дальние указатели имеют длину 4 байта, ближние — 2 байта и короткие — 1 байт. В ассемблере эти указатели сопоставляются с метками, обозначающими точку перехода. А их тип определяется по расстоянию в байтах между командой перехода и точкой перехода.

Вне зависимости от дальности различают переходы **безусловные** и **условные**. Безусловные передачи управления производятся в любом случае, независимо ни от каких условий, а условные — только при выполнении заданных в команде условий.

Безусловные переходы

К группе команд безусловной передачи управления относятся команды собственно безусловного перехода, команды перехода к подпрограмме (вызова подпрограмм) и команды возврата из подпрограмм.

Команда безусловного перехода имеет уже упоминавшийся мнемокод `jmp`. Для организации безусловного перехода в регистры `cs` и `ip` (или только `ip`) загружается их новое содержимое, то есть новый указатель. При этом их старое содержимое — старый указатель — безвозвратно теряется. Это значит, что после совершения перехода восстановить точку программы, из которой он был совершен, невозможно. На рис. 4.36, *а* приведен пример выполнения безусловного перехода по команде `jmp M1`.

Вызов подпрограмм осуществляется с помощью команды с мнемокодом `call` (вызвать). В команду `call` входит метка директивы `proc` (от procedure — процедура), с которой в языке Ассемблер принято начинать подпрограммы. Заканчиваются подпрограммы директивой `endp` (от end procedure — конец процедуры), которая должна иметь ту же самую метку, что и директива `proc`.

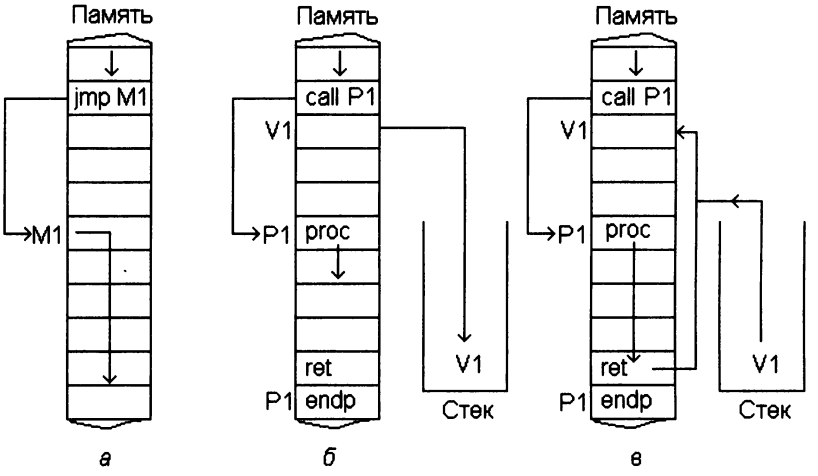


Рис. 4.36. Схемы выполнения команд безусловных передач управления

Обращение к подпрограмме происходит следующим образом. Вначале старое значение указателя из регистров `cs:ip` помещается в стек (рис. 4.36, *б*). Отметим, что в момент выполнения команды `call` регистры `cs:ip` содержат адрес команды, которая расположена непосредственно за этой командой вызова подпрограммы. На рис. 4.36, *б* эта точка программы отмечена меткой V1. Затем из команды вызова выбирается новый указатель (в примере на рисунке это указатель на метку P1 начала подпрограммы), который загружается в регистры `cs:ip`. Тем самым обеспечивается переход к выполнению вызванной подпрограммы.

Сохранение в стеке адреса, откуда был совершен переход в подпрограмму, обеспечивает возможность при необходимости вернуться в точку вызова. Чтобы такой возврат произошел, в подпрограмму должна быть включена команда `ret` (от return — возврат), схема выполнения которой показана на рис. 4.36, *в*. По команде `ret` производятся выборка указателя из вершины стека и последующая его загрузка в пару регистров `cs:ip`. В примере на рис. 4.36 из стека выбирается

указатель, соответствующий метке V1, которой отмечена команда, непосредственно следующая за командой вызова `call P1`.

Использование стека для хранения последовательности адресов возвратов позволяет правильно организовать вложенные обращения к подпрограммам. На рис. 4.37 изображена схема выполнения вложенных вызовов, когда из основной программы вызывается подпрограмма P1, а из нее — другая подпрограмма P2. Во время выполнения первого вызова по команде `call P1` (рис. 4.37, *а*) в стек попадает адрес точки возврата V1 и управление передается в начало подпрограммы P1. Внутри подпрограммы P1 вызывается другая подпрограмма P2. По команде `call P2` в стек попадает адрес второй точки возврата V2 и происходит переход в начало второй подпрограммы P 2 (рис. 4.37, *б*).

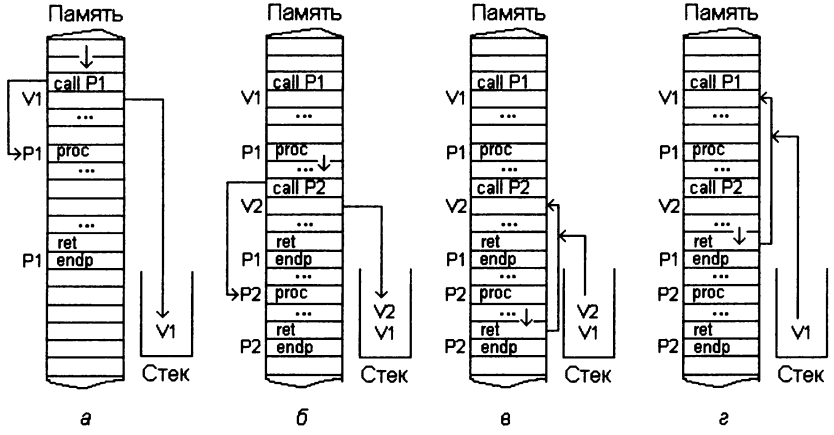


Рис. 4.37. Схема выполнения вложенных обращений к подпрограммам

Предполагается, что в теле подпрограммы P2 других вызовов нет. Поэтому в момент выполнения находящейся в теле подпрограммы P2 команды `ret` из вершины стека выбирается адрес второй точки возврата V2 (рис. 4.37, *в*) и таким образом осуществляется возвращение в первую подпрограмму P1, в ту ее точку, из которой была вызвана подпрограмма P2.

Когда закончится выполнение подпрограммы P1 и начнется выполнение ее команды `ret` (рис. 4.37, *г*), из вершины стека окажется выбранным адрес первой точки возврата V1 и, следовательно, осуществится возврат в основную программу.

Обсуждаемая схема обеспечивает корректную обработку вызовов подпрограмм с произвольной глубиной вложенности. Если быть точным, то глубина вложенности вызовов ограничивается только объемом стека.

Условные переходы

Команды условного перехода анализируют состояние определенных флажков процессора, и если это состояние удовлетворяет заданному в команде условию, то происходит передача управления по находящемуся в команде указателю.

В противном случае новое значение указателя не загружается и, следовательно, выполняется команда, следующая по порядку за командой условного перехода, то есть сохраняется линейный порядок выполнения программы.

В системе команд i8086 предусмотрено около трех десятков различных команд условного перехода, которые анализируют различные флажки и различное их состояние. Примеры некоторых команд условных переходов приведены в табл. 4.22.

Таблица 4.22. Команды условных переходов

Название	Мнемокод	Условие	Флажки
Если равно/Если нуль	`je/jz`	`=/0`	`zf = 1`
Если не равно/Если не нуль	`jne/jnz`	$\neq/\neq 0$	`zf = 0`
Если меньше/Если не больше или равно	`jl/jnge`	$</\neg \geq$	`sf` $\oplus$ `of = 1`
Если не меньше/Если больше или равно	`jnl/jge`	$\neg </\geq$	`sf` $\oplus$ `of = 0`
Если больше/Если не меньше или равно	`jg/jnle`	$>/\neg\leq$	`(sf` $\oplus$ `of)` $\vee$ `zf = 0`
Если не меньше/Если больше или равно	`jng/jle`	$\neg >/\leq$	`(sf` $\oplus$ `of)` $\vee$ `zf = 1`
Если выше/Если не ниже или равны	`ja/jnbe`		`cf` $\vee$ `zf = 0`
Если ниже/Если не выше или равны	`jb/jnae`		`cf = 1`

$\oplus$ — обозначает логическую операцию «Исключающее ИЛИ».

Буквы, входящие в мнемокоды команд, имеют следующий смысл: `j` — jump (переход), `e` — equal (равно), `z` — zero (нуль), `n` — not (не), `g` — greater (больше), `l` — less (меньше), `a` — above (выше), `b` — below (ниже). Зная эти обозначения, по мнемокоду легко понять включенное в команду условие. Например, команда `jne` имеет смысл: «переход, если не равно».

Обычно одна и та же команда имеет два эквивалентных названия и обозначения. В табл. 4.22 эти названия и обозначения разделены наклонной чертой «/». Например, известно, что отношение «меньше» можно заменить эквивалентным отношением «не больше или равно». Поэтому команда перехода с названием «переход, если меньше» (имеется в виду меньше нуля) и мнемокодом `jl` (от jump if less) может быть также названа «переход, если не больше или равно») и обозначена другим мнемокодом `jnge` (от jump if not greater or equal). Из аналогичных соотношений образованы мнемокоды и названия остальных команд условного перехода.

По третьей строке табл. 4.22 видно, что обсуждаемой команде соответствуют такие состояния флажков процессора, для которых `sf` $\oplus$ `of = 1`. По определению логической операции «Исключающее ИЛИ», это соотношение имеет место, если `sf` и `of` имеют разные значения, то есть если `sf = 1` и при этом `of = 0` или же `sf = 0`, но `of = 1`. Как известно, флажок `sf` принимает значение 1 в том случае, если в результате выполнения текущей команды получился код, который можно трактовать как код отрицательного числа (знаковый бит поля занят единицей), то есть как код числа, меньшего, чем нуль. Однако при этом требуется, чтобы флажок `of` был равен нулю, то есть чтобы он не показывал выход из диапазона знакового представления чисел. Собственно говоря, именно эта ситуация

подразумевается при использовании команды `jl`. Кроме того, этой команде соответствует ситуация, когда `of` = 1 и при этом `sf` = 0, то есть имеет место знаковое переполнение. В связи с этим можно утверждать, что указанное в таблице выражение `sf` ⊕ of = 1 более точно отслеживает ситуацию, что, на первый взгляд, более простой анализ значения флажка `sf` = 1. Это замечание касается всех команд условного перехода, реагирующих на комбинацию флажков `sf` ⊕ of.

Итак, если при выполнении рассматриваемой команды `jl/jnge` состояние флажков `sf` и `of` удовлетворяют указанному условию, то входящий в команду указатель заменяет текущее содержимое регистров `cs:ip` и тем самым организуется переход в другую точку команды. В противном случае содержимое этих регистров не изменяется, и следовательно, выполняется следующая по порядку команда. Этот принцип действия распространяется на все команды условного перехода. Разница только в проверяемых флажках и условиях, которым должны удовлетворять их состояния.

Теперь обратим внимание на две последние строки табл. 4.22, в которых используются отношения «выше» и «ниже». Чтобы понять смысл этих отношений, следует выяснить, какой из кодов 0000 0000_2 и 1111 1111_2 больше, а какой меньше. Оказывается, что ответ на этот вопрос не очевиден. Если рассматривать эти коды как знаковые, то коду 1111 1111_2, соответствует число -128_{10}, а коду 0000 0000_2 — число 0_{10}, то есть код 1111 1111_2 оказался меньше, чем код 0000 0000_2. С другой стороны, если трактовать эти коды, как беззнаковые, то коду 1111 1111_2 соответствует число 255_{10}, которое очевидно больше, чем число 0_{10}, соответствующее коду 0000 0000_2. Чтобы различать эти ситуации, не прибегая к такому детальному анализу, как раз и введены эти отношения. Итак, код 0000 0000_2 ниже, но при этом больше, чем код 1111 1111_2, и наоборот, код 1111 1111_2 выше, но меньше, чем 0000 0000_2. То есть отношения «выше» и «ниже» в названиях и мнемокодах команд условных переходов касаются соотношений между кодами, трактуемыми как беззнаковые, а отношения «меньше» и «больше» используются при знаковой трактовке кодов.

4.3.12. Организация ветвлений на машинном уровне

Чтобы понять особенности организации ветвлений на машинном уровне, рассмотрим простейший пример определения наименьшего числа из двух заданных. Обозначим исходные числа именами `R1` и `R2`, а результат назовем `R3`. Тогда на языке Паскаль это действие можно изобразить с помощью условного оператора `if R1 < R2 then R3 := R1 else R3 := R2`. Этот же алгоритм можно изобразить и в виде блок-схемы, приведенной на рис. 4.38, *а*. Ее особенностью является плоский, двухмерный характер. Блоки, описывающие различные ветви алгоритма, идут не последовательно, они параллельны друг другу. Но оперативная память компьютера имеет линейный, одномерный характер. Линейностью обладает также стандартный порядок выполнения команд программы. Следовательно, нужно найти способ отображения двухмерной структуры ветвления на одномерные структуры программы и памяти.

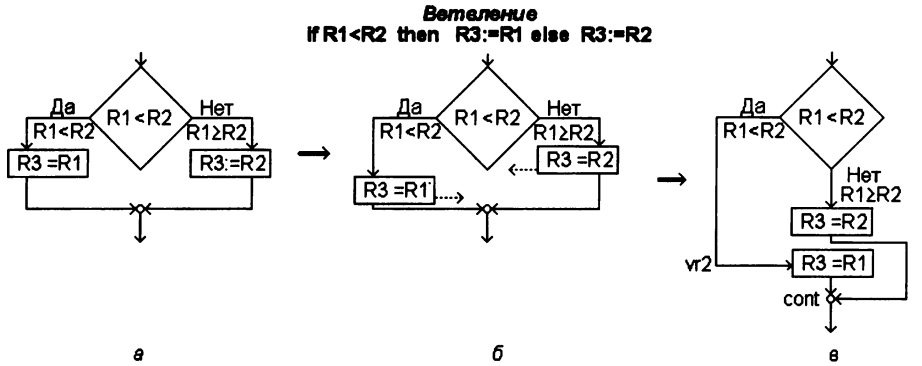


Рис. 4.38. Трансформации блок-схемы ветвления к машинному уровню реализации

Чтобы отобразить плоскую структуру блок-схемы на линейную структуру оперативной памяти, деформируем схему, оставляя неизменными отображаемые ею действия и связи. На рис. 4.38, *б* и *в* показаны основные этапы такой деформации. На первом этапе блоки разносятся по вертикали, что не влияет на описываемые действия. Затем блоки ветвей при сохранении имеющихся между ними связей как бы сдавливаются с боков и выстраиваются в линию. Итоговый вид схемы изображен на рис. 4.38, *в*.

Сравнивая получившиеся варианты блок-схем, приходим к выводу, что по смыслу выполняемых действий они эквивалентны друг другу. Но последний вариант блок-схемы, как показано далее, позволяет с помощью команд условного и безусловного перехода отобразить плоскую структуру схемы на линейную оперативную память.

Для программной реализации полученного алгоритма с помощью директив `dw` в сегменте данных выделим двухбайтовые рабочие поля для хранения значений переменных `R1`, `R2` и `R3`. Блок-схема предписывает следующий порядок действий, реализующих ветвление. Вначале нужно сравнить значения величин `R1` и `R2` в отношении «меньше». Поскольку оба операнда находятся в полях памяти, необходимо один из них переслать в регистр процессора. Перешлем, например, значение из поля `R1` в регистр `ax`. Теперь для сравнения величин `R1` и `R2` можно использовать команду `cmp ax,R2`, которая, как известно, не изменяет значения операндов, а только формирует соответствующие результату сравнения значения флажков. Если окажется, что `R1 - R2 < 0`, то есть `R1 < R2`, то флажок `sf = 1`. В этом случае нужно организовать переход к ветви, в которой `R3 := R1`. На значение флажка `sf = 1` реагирует команда `jl` (предполагается, что знакового переполнения не происходит и `of = 0`), а действия, которые в этом случае нужно выполнить, обозначены меткой `vr2`. Следовательно, команда, которая организует требуемый в ветвлении переход, имеет вид `jl vr2`.

Если окажется, что `R1 - R2 ≥ 0`, то есть определенное в команде перехода условие не выполнено, то, согласно схеме выполнения команды `jl`, управление переходит к следующей по порядку команде программы. В соответствии с блок-схемой в этом случае нужно выполнить действие `R3 := R2`. И вновь для промежуточного

хранения требуется выбрать регистр процессора. Пусть это будет регистр bx. Перешлем в него нужное значение командой mov bx,R3. Таким образом, соответствующая не выполненному условию ветвь реализована. Теперь необходимо попасть в общую точку программы, которая обозначена меткой cont. Поскольку переход нужно выполнять в любом случае, его следует реализовать с помощью команды безусловного перехода jmp cont.

Теперь займемся второй ветвью программы, которая должна иметь метку vr2 и выполнять действие R3 := R1. Это действие может быть реализовано командой пересылки либо из поля R1, либо из регистра ax. Целесообразно осуществить пересылку в тот же самый регистр, в который попал результат в уже реализованной ветви. Таким образом, приходим к команде вида vr2 mov bx, R1. Отметим, что эта команда должна располагаться в программе, а следовательно, и в оперативной памяти *непосредственно* за командой безусловного перехода.

Согласно блок-схеме, организация ветвления завершается действиями, общими для обеих ветвей, причем они должны начинаться командой с меткой cont. В данном случае общее действие состоит в пересылке результата из регистра bx в подготовленное для него поле R3.

Фрагменты полученной в результате проведенных рассуждений программы изображены на рис. 4.39, *слева*. Следует понимать, что показывающие выполняемые переходы стрелки в реальных программах отсутствуют. На рисунке они приведены, чтобы проследить за общими элементами в структуре программы и изображенной справа ее блок-схемой.

```
dtsg    segment   'data'
R1      dw    ?   ; 16-ричные данные
R2      dw    ?
R3      dw    ?
dtsg    ends
-------------------------------------------------
cdsg    segment   'code'
        ...
        mov   ax,R1  ;[ax]=R1
        cmp   ax,R2  ;если R1-R2<0, то sf=1
R1<R2   jl    vr2    ;переход если меньше на метку VR2
          R1≥R2
        mov   bx,R2  ;первая ветвь, пересылка R2 в bx
        jmp   cont   ;переход в точку соединения ветвей
vr2---> mov   bx,R1  ;вторая ветвь, пересылка R1 в bx
cont    mov   R3,bx  ;общий участок, результат в R3
cdsg    ends
```

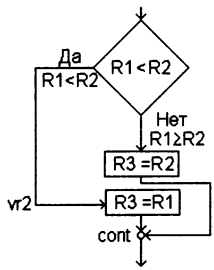


Рис. 4.39. Фрагмент программной реализации ветвления

В заключение обсуждения этого примера отметим важную роль, которую играет в организации ветвления команда безусловной передачи управления jmp cont.

Она как бы отделяет одну ветвь программы от другой. Предположим, что программист пропустил эту команду. В результате допущенной ошибки после выполнения команды `mov bx, R2` сразу же начнется выполнение команды `mov bx, R1`, которая изменит сформированное предыдущей командой значение в регистре `bx`. Поэтому результатом выполнения ошибочной программы в любом случае окажется число из поля `R1`.

Теперь сделаем общий вывод по реализации ветвления на машинном уровне. Вначале нужно организовать вычисление значения выражения, от которого зависит деление алгоритма на ветви. В рассмотренном примере таким выражением является `R1 - R2`. В это время формируются значения флажков процессора, ориентируясь на которые, выбирается наиболее подходящая команда условного перехода. В этой команде указывается метка ветви, соответствующей выполненному условию ветвления. Непосредственно за командой условного перехода организуются действия ветви, соответствующей не выполненному условию ветвлений. Завершается эта ветвь командой безусловного перехода на общий участок программы. За командой безусловного перехода организуются действия ветви, соответствующей выполненному условию ветвления. Начинаются эти действия командой с меткой, указанной в команде условного перехода. Сразу после окончания второй ветви следует разместить действия, общие для обеих ветвей. Они должны начинаться командой с меткой, указанной в команде безусловного перехода.

4.3.13. Организация циклов на машинном уровне

В качестве примера для реализации цикла на машинном уровне рассмотрим простую задачу вычисления суммы

$$S = \sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n,$$

где n — задаваемое с помощью ввода количество слагаемых. Характерной чертой этой суммы, упрощающей написание программы, является совпадение значения слагаемого с его номером.

Для хранения количества слагаемых предлагается зарезервировать однобайтовое поле, что позволит решать такую задачу для любых $n \le 255_{10}$. Для хранения результата зарезервируем поле директивой `dw`, так как для достаточно больших n значение искомой суммы наверняка выйдет за пределы одного байта.

Воспользуемся стандартным алгоритмом накопления суммы. Для промежуточного хранения накапливаемой суммы лучше использовать не поле оперативной памяти, а один из регистров, например регистр `ax`. Однобайтовый номер текущего слагаемого может быть размещен в любом восьмибитовом регистре (кроме уже занятых одновременно с `ax` регистров `ah` и `al`). Пусть это будет, например, регистр `cl`.

Подготовка к накоплению суммы сводится к закреплению за величиной `s` начального нулевого значения `s := 0`. Это можно организовать, например, командой очистки регистра `sub ax, ax`. Перебор слагаемых целесообразно начать

с последнего слагаемого, с номером n: `i := n`. Для этого в регистр `cl` нужно переслать из поля `n` номер последнего слагаемого: `mov cl, n`.

Если планируется работа с любым 8-битовым регистром, необходимо постоянно помнить о регистре, который дополняет его до 16-битового. В данном случае для накопления суммы очередное слагаемое, которым является содержимое регистра `cl`, нужно добавлять к содержимому регистра `ax`, в котором накапливается сумма. Подходящая на первый взгляд команда сложения `add ax, cl` ошибочна, так как операнды имеют различную длину. Поэтому добавлять слагаемые придется командой `add ax, cx`. Но в этом случае в накапливаемую сумму может попасть возможно ненулевой код из регистра `ch`. Поэтому перед началом накопления суммы этот регистр необходимо очистить, например, командой `sub ch, ch`.

Так как к команде сложения нужно будет неоднократно возвращаться, она обязана иметь метку: `cont add ax, cx`. После добавления текущего слагаемого нужно перейти к предыдущему (так как перебор начался с последнего слагаемого). В рассматриваемом случае для этого достаточно уменьшить номер слагаемого на единицу, например, командой декремента `dec cl`.

Теперь нужно проверить, завершен перебор слагаемых или нет, и, если он не завершен, возвратиться к команде с меткой `cont`, добавляющей очередное слагаемое. Условием продолжения суммирования в рассматриваемой задаче, очевидно, является неравенство $i > 0$ — слагаемые добавляются до тех пор, пока номер очередного слагаемого больше нуля. Из имеющихся команд условной передачи управления нужно выбрать такую команду, которая обеспечивает переход по условию $i > 0$ или, что в данном случае то же самое, `[cl] > 0`. Наиболее подходящей, по-видимому, является команда `jg`, которая обеспечивает переход, если значения флажков процессора соответствуют положительному результату в предшествующей команде декремента `dec cl`. В выбранной команде условного перехода должна быть указана метка команды, с которой начинается цикл: `jg cont`.

```
dtsg    segment   'data'
n       db     200     ;количество слагаемых
s       dw     ?       ;искомая сумма
dtsg    ends
cdsg    segment   'code'
        …
        sub    ax,ax  ;s:=0
        mov    cl,n   ;i:=n
        sub    ch,ch
cont    add    ax,cx  ;s:=s+i
        dec    cl     ;i:=i-1
        jg     cont   ;i>0 ?
        …
        mov    s,ax
cdsg    ends
```

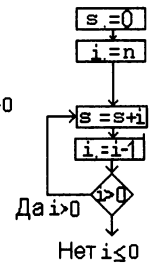


Рис. 4.40. Фрагмент программной реализации цикла

Каждый раз при очередном выполнении тела цикла эта команда передает управление на начальную команду цикла и очередное слагаемое добавляется к накапливаемому результату. После добавления последнего слагаемого по команде декремента содержимое регистра `cl` станет равным нулю, и тогда команда условного перехода передаст управление следующей по порядку команде, тем самым завершив выполнение цикла. Объединяя вместе все эти команды, получим фрагмент программы, приведенный на рис. 4.40.

4.3.14. Работа с массивами

В качестве примера обработки данных с более сложной структурой обсудим некоторые приемы работы с массивами. Для этого разработаем программу вычисления суммы

$$S = \sum_{i=1}^{100} a_i.$$

Как выяснится немного позже, нумеровать слагаемые удобнее начиная с нуля. Поэтому преобразуем искомую сумму к эквивалентному виду:

$$S = \sum_{i=0}^{99} a_i.$$

В этой сумме, в отличие от предыдущего случая, слагаемые могут быть произвольными целыми числами. Будем считать величины a_i элементами массива, которые задаются с помощью ввода в процессе выполнения программы. Организацию ввода, как и во всех предыдущих задачах, рассматривать не будем, а сосредоточим внимание на вопросах работы с массивом на уровне машинного языка.

Прежде всего нужно выделить место в памяти для хранения всех элементов массива. Для простоты будем считать, что каждое слагаемое может быть представлено беззнаковым однобайтовым кодом. Это ограничение означает, что рассматриваются только слагаемые, удовлетворяющие неравенствам $0 \le a_i \le 255$, $\forall i = 0, 1, 2, \ldots, 99$. Элементы массива `a` целесообразно разместить в последовательности из 100 подряд расположенных байтов. Это можно сделать с помощью директивы `a db 100 dup(?)`, которая обеспечивает приведенную на рис. 4.41 схему размещения элементов массива в оперативной памяти.

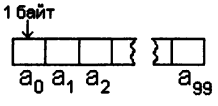


Рис. 4.41. Схема хранения однобайтовых элементов массива в оперативной памяти

Для доступа к отдельным элементам массива предусмотрена обсуждавшаяся ранее косвенная адресация с индексированием и смещением. Для определенности выберем индексирование через регистр `di`. Если осуществлять перебор значений в регистре `di` от 0 до 99, то с помощью записанного в ассемблерном формате адреса `a[di]` можно по очереди выбрать каждое из слагаемых, занимающих поле `a`. В самом деле, если в регистре `di` находится нулевое значение, то `a[0]`

представляет собой внутрисегментное смещение, определяющее адрес поля a, то есть адрес нулевого слагаемого. Каждый последующий инкремент индекса di в операнде a[di] обеспечивает адресацию следующего байта поля a, то есть следующего слагаемого. Эти соображения, собственно говоря, и лежат в основе сделанного ранее перехода от суммирования в пределах от 1 до 100 к суммированию в пределах от 0 до 99.

Накопление суммы можно осуществить с помощью схемы, обсуждавшейся в предыдущем примере (см. рис. 4.40). Однако в учебных целях рассмотрим вариант построения цикла с возрастающим значением индекса, который, согласно формуле, должен изменяться от 0 до 99.

В соответствии с предыдущими рассуждениями для накопления суммы выберем регистр ax, а для хранения индекса выделим регистр di. Тогда подготовка к циклу заключается в очистке регистров ax и di. Способы такой очистки обсуждались ранее: sub ax, ax и sub di, si. Можно использовать и другие варианты, например mov ax, 0.

В команде добавления очередного слагаемого вместо содержащего это слагаемое регистра cx следует указать адрес поля памяти, в котором находится слагаемое. Ранее мы выяснили, что любое текущее слагаемое следует адресовать операндом a[di]. Таким образом, приходим к команде вида cont add ax, a[di]. Отметим, что адрес a[di] при [di] = 0 определяет нулевое слагаемое a_0. Поэтому в результате первого выполнения этой команды в регистр ax попадает нулевое слагаемое.

Для огранизации переходов от текущего слагаемого к следующему нужно увеличивать на единицу номер слагаемого. Это можно сделать командой inc di.

Следующий этап связан с проверкой на завершение цикла. Известно, что последним значением индекса должно быть число 99. Сравним содержимое регистра di и число 99 командой cmp di, 99. Если разность i - 99 ≤ 0, значит, еще не все слагаемые рассмотрены и нужно продолжить цикл. Для проверки такого условия и перехода на начало цикла можно использовать команду jle cont.

Объединяя выписанные ранее команды, получим приведенный в листинге 4.4 фрагмент программы, которая решает поставленную задачу.

Листинг 4.4. Обработка массива

```
dtsg    segment   'data'
a       db         100 dup(?)
s       dw         ?
dtsg    ends
cdsg    segment   'code'
        ...
        sub       ax,ax       ;s:=0 можно mov ax,0
        sub       di,di       ;i:=0 можно mov di,0
cont    add       ax,a[di]    ;s:=s+a_i
        inc       di          ;i:=i+1
        cmp       di,99       ;i-99 <= 0 ?
        jle       cont        ;если <= 0,то переход на cont
```

```
        mov     s,ax
        ...
cdsg    ends
```

Мысленно проверим ход накопления суммы. После первого возврата к команде суммирования `[di] = 1`, следовательно, адрес `a[di]` определяет слагаемое a_1. Аналогично, после второго возвращения `[di] = 2`, и тот же самый адрес определяет a_2. Вообще после возвращения с номером i к сумме добавляется слагаемое ai. Последний возврат произойдет, когда `[di] = 99`, следовательно, к накапливаемой сумме добавится последнее слагаемое a_{99}. После этого инкремент регистра `di` нарушит условие, по которому осуществляется переход командой `jle cont`. Управление перейдет к следующей команде `mov s, ax`, и на этом выполнение цикла завершится.

Отметим, что для перебора слагаемых может быть использована и чисто индексная адресация. Но тогда перед началом перебора слагаемых в регистр `di` необходимо загрузить адрес нулевого слагаемого. Это можно сделать, например, обсуждавшейся в 4.3.5 командой `mov di, offset a`. Теперь в команде суммирования нужно задавать адрес текущего слагаемого в виде `byte ptr [di]`, то есть с указанием признака длины операнда.

При таком подходе возникает трудность, связанная с определением момента завершения цикла. В предыдущем примере значения в `di` изменялись от 0 до 99, что позволяло просто сформулировать условие завершения цикла. В предложенном варианте значения в `di` изменяются от адреса поля `a` до этого же адреса, увеличенного на 99. Выделение из содержимого регистра `di` текущего значения индекса сопряжено с техническими трудностями. Простой вариант решения проблемы состоит в использовании дополнительного счетчика количества повторений тела цикла. В качестве такого счетчика удобно выбрать регистр `cx` или `cl`, так как это их индивидуальная специализация. При этом цикл можно организовать так же, как на рис. 4.40:

```
        sub   ax,ax              ;s:=0
        mov   di, offset a       ;адрес a0 в регистр di
        mov   cl,99              ;инициализация счетчика
cont    add   ax, byte ptr[di]   ;s:=s+a1
        inc   di                 ;переход к следующему слагаемому
        dec   cl                 ;изменение счетчика
        jg    cont               ;переход на cont
        mov   s,ax
```

В Ассемблере, как и в языках высокого уровня, имеются определенные средства автоматизации построения циклов. В частности, в нем предусмотрена команда `loop`, которая похожа на цикл `repeat until` языка Паскаль. Работа этой команды основана на специализации регистра `cx` как счетчика количества повторений тела цикла. Чтобы использовать эту команду перед циклом, необходимо записать в регистр `cx` целое число, которое определяет необходимое количество повторений тела цикла. В конце цикла ставится команда `loop` с меткой на его начало. Эта команда при каждом выполнении тела цикла уменьшает значение в регистре

cx на единицу и сравнивает его текущее значение с нулем. Если содержимое регистра cx больше нуля, то управление передается на метку, указанную в команде:

```
        ...
        sub     ax,ax       ;s:=0
        mov     di,0        ;i:=0
        mov     cx,100      ;[cx]=100,количество повторений
cont    add     ax,a[di]    ;s:=s+a[i]
        inc     di          ;i:=i+1
        loop    cont        ;организация цикла по счетчику cx
        mov     s,ax
        ...
```

В приведенном фрагменте цикл, реализующий те же самые вычисления, что в предыдущих листингах, базируется на использовании команды loop. Удобство этого варианта в том, что программист избавлен от организации проверки завершения цикла. Он просто записывает в конце цикла команду loop с меткой, начинающей цикл.

В заключение обсуждения архитектуры процессора i8086 отметим, что обсуждавшимися ранее командами не исчерпывается его система команд. Рассматривались только команды, необходимые для иллюстрации изучаемых вопросов архитектуры компьютера. В частности, совершенно не затронуты логические команды, цепочечные команды и некоторые другие группы команд.

Заметим также, что все обсуждавшиеся команды оперируют целочисленными данными в формате с фиксированной точкой. Это далеко не случайно, так как процессор i8086 принципиально не работает с вещественными данными в формате с плавающей точкой. Если возникает необходимость в таких операциях, то операционная или программная система обращается к включенным в ее состав подпрограммам, которые эмулируют выполнение действий над вещественными данными.

ВНИМАНИЕ

Эмуляцией называется реализация действий одних аппаратных или программных систем с помощью других систем, в которых такие действия не предусмотрены.

Другими словами, процессор i8086 реализует действия над данными в формате с плавающей точкой с помощью команд, оперирующих с данными в формате с фиксированной точкой. Отметим, что это довольно медленная процедура.

Предусмотрен и более эффективный вариант организации вычислений с плавающей точкой. Для его реализации в состав компьютера включается так называемый **математический сопроцессор** i8087 (дополнительный процессор), или **FPU** (от Float Point Unit — устройство с плавающее точкой), специализирующийся на обработке данных с плавающей точкой. Он осуществляет вычисления над такими данными значительно быстрее, чем основной процессор эмулирует эти действия.

Упрощенно схема взаимодействия основного процессора и сопроцессора выглядит следующим образом. Если в программе встречаются действия над вещественными данными, основной процессор передает всю необходимую для их выполнения

информацию сопроцессору, а сам продолжает выполнение действий, не связанных с такими данными, переключается на выполнение другой программы или переходит в состояние ожидания. Сопроцессор, выполнив заданную обработку вещественных данных, сообщает основному процессору об этом и переключается на выполнение других вычислений или переходит в состояние ожидания. Видно, что схема взаимодействия процессора и сопроцессора похожа на реализованную с помощью прерываний схему обмена.

Полное изучение системы команд процессора i8086 не является целью данной главы учебника. Включенного в главу материала достаточно, чтобы понять сущность работы компьютера на уровне машинных кодов, а при необходимости по дополнительной литературе полностью освоить программирование на языке Ассемблер.

Дополнительный материал по обсуждавшимся в главе вопросам можно найти в изданиях [1], [13], [25], [33].

Контрольные вопросы и упражнения

1. Опишите назначение и основные свойства оперативной памяти компьютера.
2. Чем адрес байта отличается от его номера?
3. Сравните способы нумерации байтов и битов внутри поля памяти.
4. Поясните смысл терминов «стандартное поле», «нестандартное поле».
5. Охарактеризуйте все стандартные поля памяти.
6. Дайте определение понятий «система команд процессора», «машинная команда», «программный код».
7. Какими техническими параметрами характеризуются процессоры?
8. Какие функции выполняет шина? Опишите ее структуру.
9. Поясните смысл терминов «слот», «сокет», «порт».
10. Что называется адресным пространством? Как определяется его объем?
11. Нарисуйте схему компьютера с шинной архитектурой.
12. Для чего нужен арбитраж шины?
13. Как определяется пропускная способность шины?
14. Чем принципиально отличается внешняя память от оперативной?
15. Поясните смысл терминов «загрузка», «передача управления».
16. Охарактеризуйте назначение известных вам периферийных устройств компьютера.
17. Опишите программную модель оперативной памяти.
18. Для чего нужно выравнивать данные по границам слов памяти?
19. Сформулируйте принцип обратной записи.
20. Для чего потребовалось вводить сегментацию оперативной памяти?
21. Что такое сегмент памяти? Чем он характеризуется?

22. Опишите организацию сегментной адресации.
23. Что представляет собой полный указатель и как он задается?
24. Опишите программную модель процессора.
25. Что представляет собой регистровый уровень памяти?
26. Сформулируйте основные различия между регистровым уровнем памяти и оперативной памятью.
27. Перечислите основные группы регистров процессора, укажите их назначение и особенности структуры.
28. Охарактеризуйте специализацию каждого из регистров общего назначения.
29. Охарактеризуйте особенности использования указательных и индексных регистров.
30. Охарактеризуйте особенности использования сегментных регистров.
31. Опишите механизм формирования физического адреса в процессоре.
32. Опишите структуру регистра флажков.
33. Какую роль играют флажки `zf` и `sf` в организации вычислений?
34. Как по значениям флажков `cf` и `of` определить правильность выполнения операций над беззнаковыми и знаковыми целыми числами?
35. Опишите общую структуру машинной команды.
36. Что определяет адресность машинной команды? Опишите особенности машинных команд различной адресности.
37. Охарактеризуйте применяющиеся в машинных командах способы адресации.
38. Чем отличается исполнительный адрес от адреса команды?
39. Сравните между собой машинный и ассемблерный форматы команды.
40. Для чего нужна трансляция программы?
41. Опишите структуру и приведите примеры безадресных команд процессора i8086.
42. Опишите структуру и приведите примеры регистровой модификации одноадресных команд процессора i8086.
43. Опишите структуру общей модификации одноадресных команд процессора i8086.
44. Какую роль в выполнении машинных команд играет параметр `mod`?
45. Какую роль в выполнении машинных команд играет параметр `r/m`?
46. Опишите порядок формирования эффективного адреса по заданным в машинной команде элементам.
47. Как можно задать косвенную адресацию с базированием? Как можно задать косвенную адресацию с индексированием?
48. Как можно задать косвенную адресацию с базированием, индексированием и смещением?

49. Код операции команды записи в стек содержимого некоторого регистра 01010_2. Запишите машинный эквивалент команд `push cx`, `push ax`.
50. Каким может быть код команды выборки из стека в регистр? Запишите машинный эквивалент команд `pop dx`, `pop ax`. Почему не допускается использование в команде `pop` регистра `cs` в качестве получателя слова, выбранного из вершины стека?
51. Как определяется длина операндов в ассемблерном формате команд?
52. Какую роль играет префикс замены сегмента в адресации операндов?
53. Опишите структуру и приведите примеры двухадресных команд процессора i8086.
54. Какие команды представлены машинными кодами 03 EA_{16}, 01 EA_{16}?
55. Запишите машинные коды команд `add ax,si`; `add si,ax`; `add ch,dl`; `add dl,ch`. Имеет ли это задание однозначный ответ?
56. Опишите структуру и приведите примеры команд с непосредственным операндом процессора i8086.
57. Опишите алгоритм исполнения машинной команды процессором компьютера.
58. Изучите способы управления отладчиком **debug**, введите и выполните в режиме трассировки описанную в 4.2.6 машинную программу.
59. Что понимается под прерыванием? Для чего используются прерывания?
60. Приведите классификацию прерываний процессора i8086.
61. Что определяет тип прерывания? Приведите примеры прерывания различных типов.
62. Опишите порядок обработки маскируемых и немаскируемых прерываний.
63. Охарактеризуйте особенности 32-битовых процессоров Intel.
64. Сравните между собой реальный и защищенный режимы работы процессоров Intel.
65. Опишите общую структуру оператора в языке Ассемблер процессора Intel.
66. Опишите директивы сегментации языка Ассемблер процессора Intel.
67. Чем отличается программный сегмент от сегмента памяти? Как они связываются друг с другом?
68. Опишите директивы определения данных языка Ассемблер процессора Intel.
69. Не используя повторителей, запишите эквивалент конструкции `3dup(5,18H, 'строка', ?, 5 dup(' '))`.
70. Как вы думаете, почему приняты существующие в Ассемблере правила записи двоичных, восьмеричных и шестнадцатеричных констант?
71. Напишите на Ассемблере эквивалент фрагмента программы на языке Паскаль `const n=2; pi=3.14159`.
72. Напишите на Ассемблере эквивалент фрагмента программы на языке Паскаль `var i,j: integer; a:array[1..3,1..4] of real`.

73. Приведите примеры команд Ассемблера с разными способами адресации операндов.
74. Приведите примеры возможных вариантов команды пересылки.
75. Какие команды работы со стеком вам известны? Опишите особенности их выполнения.
76. Опишите состав и особенности выполнения команд сложения и вычитания.
77. Опишите состав и особенности выполнения команд умножения и деления.
78. Опишите особенности организации действий над многоразрядными данными.
79. Напишите на Ассемблере программы вычисления значения выражений

$$x = \frac{2|a-b|}{3(c+d)^2}, x = \frac{a^2 - b^2}{a^2 + b^2},$$

считая, что значения величин *a*, *b*, *c*, *d* могут быть записаны в 8-битовые поля.

80. Нужно ли что-либо изменить в программе, показанной в листинге 4.2, чтобы она работала со знаковыми данными?
81. Напишите на Ассемблере программы вычисления значения выражений

$$x = f - \frac{ab + cd}{c + d - 4}, x = \frac{1 - a^3 - b^3}{1 + a^2 + b^2},$$

считая, что значения величин *a*, *b*, *c*, *d* и *f* должны быть записаны в 16-битовые поля.

82. Приведите классификацию переходов в программах.
83. Опишите состав и особенности выполнения команд безусловного перехода.
84. Опишите схему выполнения вложенных обращений к подпрограммам.
85. Опишите состав и особенности выполнения команд условного перехода.
86. Опишите общую схему организации ветвлений на машинном уровне.
87. Напишите на Ассемблере программу вычисления значения выражения

$$x = \begin{cases} 7(a+b)^3 - |c-d-2|, & a <<= b; \\ ((a+4)b-2) - |c-d+2|, & a >> b, \end{cases}$$

считая, что значения величин *a*, *b*, *c*, *d* должны быть записаны в 16-битовые поля.

88. Опишите общую схему организации циклов на машинном уровне.
89. Какие средства автоматизации организации циклов предусмотрены в Ассемблере процессоров Intel?
90. Опишите возможные варианты работы с одномерными массивами в Ассемблере процессоров Intel.
91. Напишите на Ассемблере программу вычисления значения выражения $\prod_{i=1}^{100} i^2$.

Часть II
Архитектура вычислительных систем

Во второй части учебника основное внимание уделяется изучению современного состояния архитектуры вычислительных систем, а также обзору особенностей наиболее популярных архитектурных линий.

Глава 5
Развитие архитектуры и параллелизм вычислений

Целью развития вычислительных машин всегда было (да, наверно, и будет) улучшение эффективности обработки данных, выражающееся в повышении скорости обработки и увеличении объема обрабатываемых данных. Одновременно с этим разработчики добивались повышения надежности и уменьшения стоимости компьютеров, а также обеспечения удобства и упрощения работы пользователей.

В процессе развития вычислительных систем этот комплекс требований удовлетворялся за счет применения различных физических принципов хранения и обработки информации, а также усовершенствования технологий производства аппаратуры. Это приводило к желаемому повышению надежности и быстродействия отдельных устройств, росту емкости памяти, а также к уменьшению стоимости компьютеров. Кроме того, происходило уменьшение габаритов компьютеров, расширялись области использования компьютерных технологий обработки данных, упрощалось взаимодействие человека и компьютера. Одновременно с развитием аппаратных средств совершенствовались средства и методы разработки эффективного и надежного программного обеспечения.

Вместе с тем уже довольно давно стало ясно, что рано или поздно этот путь приведет к невозможности дальнейших улучшений в рамках используемого физического принципа работы компьютера. К настоящему времени возможности увеличения производительности отдельно взятого компьютера подходят к своим естественным границам, которые определяются конечностью скорости света и некоторыми другими физическими ограничениями.

Выход был найден в широком использовании параллелизма, который, по-видимому, является единственным способом дальнейшего роста производительности вычислительных систем, базирующихся на электронных устройствах.

ВНИМАНИЕ

В информатике параллелизмом называется одновременное выполнение различными устройствами или различными частями одного и того же устройства каких-либо действий по обработке информации.

В приведенном ранее общем понятии параллелизма главным является одновременная работа нескольких устройств. При этом не уточняется характер выполняемых действий по обработке информации. Это уточнение приводит к выделению **собственно параллелизма**, когда несколько устройств одновременно выполняют различные команды одной и той же или разных программ, и **конвейеризации**, когда несколько одновременно работающих устройств последовательно выполняют одну сложную операцию и при этом на разных этапах обработки одновременно находится несколько операций.

Отметим, что конвейером (от convey — транспортировать) в промышленности называется непрерывно или периодически движущееся транспортное устройство для сборки машин, обработки какого-либо материала и т. д. с *последовательным выполнением отдельных этапов несколькими одновременно работающими исполнителями (людьми, автоматами)*. Конвейер широко используется для повышения эффективности производства, так как каждый исполнитель, специализируясь на выполнении отдельной простой операции, выполняет ее быстрее, чем это делает один исполнитель, последовательно выполняющий разные операции.

ВНИМАНИЕ

Одновременное выполнение несколькими специализированными устройствами различных этапов различных операций (команд, действий и т. д.), при котором этапы одной операции выполняются этими устройствами последовательно, называется конвейеризацией.

Для собственно параллельной работы требуется несколько универсальных устройств, которые одновременно выполняют несколько сложных действий, при этом каждое универсальное устройство выполняет все этапы одного действия. Для конвейерной обработки нужно несколько более простых специализированных устройств, также работающих одновременно, при этом каждое из них выполняет один и тот же этап разных действий.

Чтобы лучше понять разницу между собственно параллелизмом и конвейеризацией, рассмотрим бытовую аналогию. Пусть имеется бригада строительных рабочих, которой поручено построить несколько домов. Понятно, что четыре таких же бригады, работая одновременно, должны построить эти же дома в четыре раза быстрее, чем одна бригада (разумеется, при наличии для работы всех необходимых условий).

Организовать одновременную (параллельную) работу нескольких бригад можно по-разному. *Собственно параллельной* является такая организация, когда каждая бригада выполняет все работы на строительстве одного и того же дома от начала до конца. Таким образом, четыре бригады одновременно построят четыре дома.

С другой стороны, можно выделить четыре специализированные группы, одна из которых готовит фундамент, другая одновременно кладет стены другого здания, третья в это же время делает крышу на третьем, а четвертая выполняет окончательную отделку на четвертом. Тогда на строительстве дома будут последовательно заняты все четыре бригады, которые по конвейеру выполняют свои операции. Преимущество конвейерной организации по сравнению с собственно параллельной в том, что узкая специализация повышает производительность каждой бригады. При этом каждый работник может иметь более низкую квалификацию, так как ему не приходится выполнять все разновидности работ. В реальных строительных организациях всегда применяется конвейерная система.

В архитектуре компьютеров широко применяются и собственно параллелизм, и конвейеризация. В последнем случае преимущества возникают из-за упрощения устройства каждого специализированного узла, которые, следовательно, могут работать быстрее, чем универсальные, а значит, более сложные и медленные устройства.

Далее во второй части учебника кратко прослеживаются основные этапы развития архитектуры вычислительных систем, акцент делается на повышении их производительности и организации параллельных вычислений. При этом производительность (мощность) компьютера в первом приближении оценивается с помощью скорости обработки данных, которая задается либо тактовой частотой процессора, либо количеством выполняемых им за одну секунду арифметических операций.

5.1. Начальные этапы развития

Потребность в выполнении различных вычислений появилась, по-видимому, вместе с человеком, всегда желавшим упростить для себя эту «сложную» работу. Тысячелетиями для вычислений использовался счет на пальцах, с помощью камешков и разнообразных насечек. Затем появились узелковый счет в доколумбовой Америке, абак (глиняная пластинка с желобками, в которых размещались камешки) в Древнем Риме, русские счеты и другие аналогичные приспособления, целиком основанные на ручной работе человека. Очевидно, что говорить о высокой эффективности счета или параллельности в этот период не приходится.

5.1.1. Механический этап

Первой известной попыткой построения механизма, предназначенного для обработки числовой информации, является относящийся примерно к 1500 г. эскиз суммирующего устройства Леонардо да Винчи. К сожалению, построить по этому эскизу реальное счетное устройство не удалось.

Первое действующее устройство для выполнения сложения было создано в 1623 г. Вильгельмом Шиккардом. Он называл свое изобретение «суммирующими часами», так как оно было создано в единичном экземпляре на базе механических часов.

В 1641–1645 гг. Блез Паскаль разработал суммирующую машину, которая получила широкую известность и была выпущена целой серией в 50 машин, из них 8 экземпляров дошло до наших дней. В честь Паскаля назван один из популярнейших современных языков программирования.

Машина Паскаля могла выполнять только сложение и вычитание, а Готфриду Вильгельму Лейбницу в 1671–1674 гг. удалось построить *арифмометр* — машину для выполнения всех четырех арифметических операций (рис. 5.1, *а*). К заслугам Лейбница в этой области следует отнести также работы по развитию двоичной системы счисления — основного «языка» всех современных компьютеров.

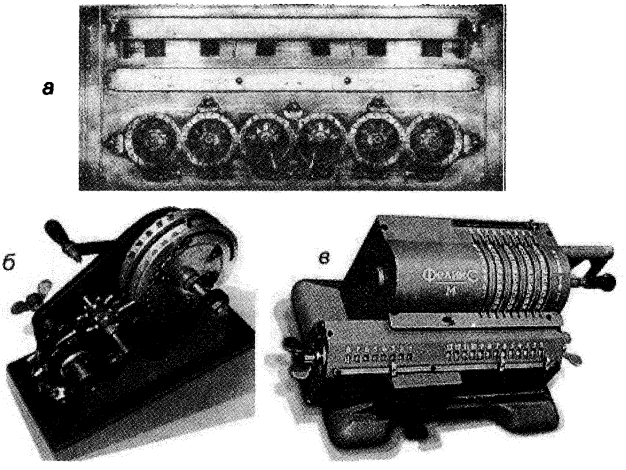


Рис. 5.1. Арифмометры: *а* — Лейбница; *б* — XIX в.; в — XX в.

Базовыми элементами, на которых основано действие машин Шиккарда, Паскаля, Лейбница и множества последовавших за ними счетных устройств, были различные валики, зубчатые колеса и т. д., совершавшие в процессе вычислений механические перемещения.

ВНИМАНИЕ

Основное устройство, на котором реализована аппаратура компьютера, называется его элементной базой.

Можно считать, что элементной базой машин этого периода является *зубчатое колесо*, которому принадлежит одна из ведущих ролей в осуществляемых на этих машинах вычислениях.

Машины Паскаля и Лейбница являются дальними предками компьютера. Но в отличие от компьютера, их работа еще не была автоматической. Эти машины, а также широко использовавшиеся в XIX и в начале XX в. арифмометры (рис. 5.1, *б* и *в*) и нынешние микрокалькуляторы характеризуются тем, что человек непосредственно участвует в вычислительном процессе на всех его этапах. В частности,

человек не только определяет последовательность выполняемых действий, но и напрямую управляет вычислениями.

5.1.2. Машины Чарльза Бэббиджа

В ходе промышленной революции XVIII–XIX вв. появились и стали широко использоваться бумажные ленты с отверстиями — *перфоленты* и листы из плотного картона с отверстиями — *перфокарты* (рис. 5.2), которые являются разновидностями долговременных носителей информации. С помощью определенных комбинаций отверстий на перфолентах и перфокартах задавался конкретный план работы различных устройств. Характерным примером такого рода устройств является ткацкий станок, изобретенный Жозефом Жаккаром во Франции в 1801–1808 гг. Наличие или отсутствие отверстия в перфокарте, управлявшей работой станка, заставляло подниматься или опускаться нить при одном ходе челнока. Станок Жаккара был первым массовым промышленным устройством, автоматически работающим по заданному плану.

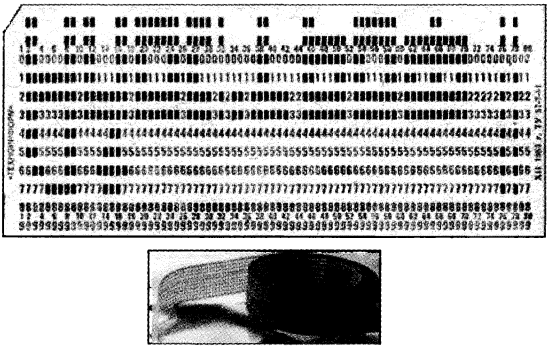


Рис. 5.2. Перфокарта и перфолента

После появления автоматических ткацких станков естественным образом должна была возникнуть мысль о том, что машине можно поручить не только изготовление тканей. По-видимому, можно попытаться поручить ей и арифметические вычисления, потребность в выполнении которых в связи с бурным ростом мореплавания была в то время очень велика. Такая мысль возникла у английского математика, профессора Кембриджского университета Чарльза Бэббиджа. В 1822 г. он опубликовал статью с описанием так называемой *разностной* машины, предназначенной для автоматического вычисления и печати таблиц математических функций, используемых в морской навигации. Позже эта машина была построена и довольно долго и успешно работала. Фактически разностная машина «умела» выполнять только один алгоритм табулирования (то есть построения таблицы значений) полиномов до 4-й степени методом конечных разностей с точностью до 15 десятичных знаков.

Затем Бэббидж начал работать над проектом машины, которую впоследствии стали называть *аналитической*. По замыслу Бэббиджа, эта машина должна была

«уметь» самостоятельно решать произвольные задачи с привлечением всех арифметических операций. Эта идея полностью исключала участие человека в вычислительном процессе, сводя его роль к подготовке необходимых числовых данных и, как и в случае с ткацким станком Жаккара, составлению плана выполнения вычислений, зафиксированного в некоторой специальной форме на перфокарте. Собственно процесс обработки информации должен был выполняться автоматически по заданной программе. Первый эскиз этой машины появился в 1834 г.

Машина Бэббиджа должна была содержать «склад», то есть память, из 1000 ячеек по 50 десятичных разрядов в каждой, вычислительное устройство — «мельницу», по терминологии Бэббиджа, а также устройство ввода обрабатываемых данных, которые планировалось хранить на перфокартах, и устройство вывода результатов на печать и на перфоратор. Программа обработки данных также должна была считываться с перфокарт.

Несмотря на несколько десятилетий работы и затраченные усилия, Бэббиджу не удалось реализовать свою идею, в основном из-за несовершенства технической базы того периода. Точность, с которой нужно было изготавливать детали этой машины, в то время была еще недостижима.

Опередивший свое время проект машины Бэббиджа содержал все основные компоненты вычислительных машин, появившихся почти через столетие после его работ. Основные идеи Бэббиджа не были забыты, они сыграли важную роль в дальнейшем развитии средств обработки информации.

Несмотря на то что аналитическая машина Бэббиджа существовала только в виде проекта, для нее была составлена первая в мире программа. В 1843 г. Ада Лавлейс, дочь английского поэта Джорджа Байрона, опубликовала работу, в которой были заложены основы современного программирования. Ею же для машины Бэббиджа была составлена программа вычисления чисел Фибоначчи. Впоследствии в ее честь был назван алгоритмический язык Ада, один из самых мощных и сложных современных языков программирования.

5.1.3. Электромеханический этап

Следующий этап в развитии средств вычислений связан с использованием так называемых **табуляторов** (от лат. tabula — доска, таблица), которые представляют собой устройства для считывания и простейших видов обработки данных, нанесенных на перфокарты. Отличительной особенностью табулятора является неизменность алгоритма обработки данных, который определяется его конструкцией.

Первый табулятор был создан Германом Холлеритом в 1887 г. Основой этого устройства являлись простейшие *электромеханические реле* (см. 3.2.1), которые составляли элементную базу вычислительных устройств в течение последующих 50–60 лет.

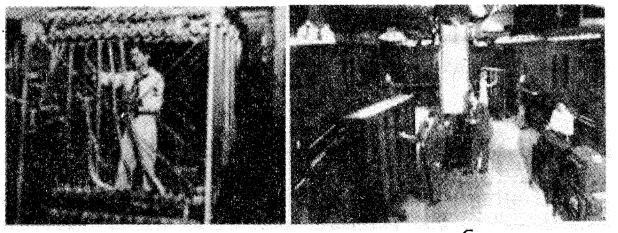
Табуляторы широко использовались для выполнения расчетов статистического характера, например для проведения переписи населения в конце XIX в. в США, Канаде, России и в некоторых других странах. Для производства табуляторов

Г. Холлерит в 1897 г. организовал фирму Tabulating Machine Company (компания по производству табуляторов), которая впоследствии преобразовалась в фирму IBM (от International Business Machines corporation — корпорация «Международные коммерческие машины»). В настоящее время эта компания является одним из мировых лидеров в сфере компьютерного производства.

В 30-х гг. XX в. в разных странах начались разработки принципиально иных устройств — программно-управляемых релейных вычислительных машин. Одна из первых таких машин под названием Z-3 была создана Конрадом Цузе в Германии в 1939–1941 гг. В ее конструкцию входило 2600 реле. Она могла «помнить» до шестидесяти четырех двадцатидвухбитовых чисел. В машине Z-3 использовалась одноадресная система команд. Сложение выполнялось за 0,3 с, а умножение — за 5 с. Предусматривались клавишный ввод данных и вывод результатов на световое табло.

Однако возможности и этой, и созданной позднее более совершенной модели Z-4 по составлению программ были довольно скромными. В частности, не было возможности осуществлять программный выбор одного из нескольких возможных вариантов действий. Это не позволяет считать Z-3 универсальной вычислительной машиной.

Полностью идеи Чарльза Бэббиджа впервые были реализованы в машине «Марк 1» (рис. 5.3, *а*), разработанной в фирме IBM под руководством Говарда Айкена в 1937–1944 гг. Машина работала в десятичной системе счисления. Ее память состояла из 72 ячеек по 23 разряда каждая. Программа работы машины задавалась с помощью различных коммутационных устройств, которые соединялись, разъединялись и переключались вручную. Весила машина 5 т, а ее работу обеспечивало устройство мощностью 5 лошадиных сил.



а *б*

Рис. 5.3. Вычислительные машины: *а* — «Марк 1»; *б* — ENIAC

«Марк 1» считается первой в мире программно-управляемой универсальной вычислительной машиной. Вместе с тем устройство для выполнения арифметических действий в ней было чисто механическим. Затем в 1947 г. была построена *полностью электромеханическая* машина «Марк 2». Она выполняла одну операцию умножения за 0,7 с.

Характерная для электромеханических машин скорость обработки данных не удовлетворяла потребностям того периода. Так, например, самая быстродействующая в мире релейная машина РВМ-1, которая была построена в 1950-х гг.

в СССР под руководством Н. И. Бессонова, выполняла операцию умножения только за 0,05 с, что соответствует выполнению 20 операций в секунду. Машина РВМ-1 была всего в 14 раз быстрее, чем «Марк 2». Дело в том, что механические перемещения — неотъемлемая часть реализации вычислительных операций в механических и электромеханических машинах — существенно ограничивали их быстродействие. Только полностью электронные, то есть исключающие механические перемещения в процессе вычислений (и, следовательно, безынерционные) устройства могли решить проблему быстродействия вычислительных машин.

5.1.4. Начало электронного этапа

Начало важнейшего на сегодняшний день **электронного** этапа в развитии средств обработки информации относится к 40-м гг. XX в. Элементной базой вычислительных систем в этот период стал *триггер* (см. 3.3), который может рассматриваться как *электронное реле* — функциональный эквивалент электромеханического реле. Триггер был изобретен М. Бонч-Бруевичем в 1913 г. на базе разработанного в 1906 г. триода — электровакуумной лампы накаливания с тремя выходными контактами.

В 1937–1942 гг. в США под руководством Дж. Атанасова и К. Берри была сконструирована первая полностью электронная машина ABC (от Atanasoff — Berry Computer), содержавшая около 600 электронных ламп накаливания. Но эта машина могла выполнять только операции сложения и вычитания и, к сожалению, так и не стала действующей. Характерными ее отличиями были применение двоичной системы счисления, а также периодически подзаряжаемые емкостные электрические элементы — конденсаторы, используемые для хранения данных. Этот принцип лежит в основе так называемой *динамической* разновидности современной оперативной памяти (см. 8.1).

Первой в мире работающей электронной цифровой машиной стал специализированный компьютер COLOSSUS, который с 1943 г. использовался англичанами для дешифровки радиосообщений, пересылаемых на немецкие подводные лодки. В создании этого компьютера принимал участие математик Алан Тьюринг, внесший значительный вклад в теоретическую информатику. В конструкцию машины входило 2000 электронных ламп.

Первая *программно-управляемая универсальная электронная* вычислительная машина была разработана в 1943–1945 гг. в Пенсильванском университете США под руководством Д. Моучли и Д. Эккерта. Эта машина называлась ENIAC (Electronic Numerical Integrator And Computer — электронно-цифровой интегратор и вычислитель) (рис. 5.3, *б*). Компьютер весил 30 т, его высота равнялась 6 м, ширина — 4 м, а общая площадь составляла 120 м^2. Машина состояла из 18 000 электронных ламп накаливания. Процессор машины содержал 20 регистров, способных хранить десятиразрядные десятичные числа. Он выполнял примерно 5000 арифметических операций в секунду (сравните с 20 операциями в секунду, выполняемыми электромеханической машиной РВМ-1). Программа для машины ENIAC задавалась вручную с помощью механических переключателей и гибких

кабелей со штекерами, вставляемыми в нужные разъемы. Поэтому любые изменения в программе требовали много сил и времени.

В 1944 г. Д. Моучли и Д. Эккерт включились в разработку машины EDVAC (от Electronic Discrete Automatic Variable Computer — параметризуемый автоматический электронный вычислитель дискретного действия), работа над которой продолжалась до 1952 г. Машина работала в двоичной системе счисления, ее память содержала свыше 5000 ячеек по 44 бита каждая. Память была реализована на так называемых ртутных линиях задержки. Система команд у машины была четырехадресной. Сложение выполнялось за 0,001 с, а умножение — за 0,002 с. Отличительной особенностью этой машины было размещение программы внутри машины, в ее оперативной памяти.

5.2. Архитектура фон Неймана

Участник разработок машин ENIAC и EDVAC выдающийся математик Джон фон Нейман, анализируя в подготовленном им с соавторами отчете «Предварительный доклад о машине EDVAC» работу первых компьютеров, фактически заложил основы архитектуры, которая в той или иной форме до сих пор используется в подавляющем большинстве вычислительных систем. Базовые принципы, выдвинутые фон Нейманом:

- использование для кодирования программ и данных двоичной системы счисления;
- применение в конструкции машины электронной элементной базы;
- организация памяти в виде линейного адресного пространства;
- хранение выполняющейся программы и обрабатываемых данных внутри машины, в ее электронных схемах памяти, а не вне ее — на перфокартах, перфолентах или разъемах со штекерами;
- организация последовательного (естественного) порядка выполнения команд программы, что приводит к отказу от четвертого адреса в командах;
- организация параллельной, то есть одновременной обработки разрядов кодов данных;
- организация параллельной передачи по внутренним линиям компьютера всех разрядов кода.

Джон фон Нейман разработал собственный проект машины IAS (от Immediate Address Storage — память с прямой адресацией), которую сейчас принято называть **фон-неймановской вычислительной машиной**, а ее архитектура считается классической **фон-неймановской архитектурой**.

Память в машине фон Неймана, схема которой изображена на рис. 5.4, состоит из 4096 ячеек по 40 битов каждая. В одной ячейке размещается один 40-битовый код целого числа, или две 20-битовые команды. Система команд в машины IAS одноадресная. Код операции занимает 8 битов, а операнд — остальные 12 битов.

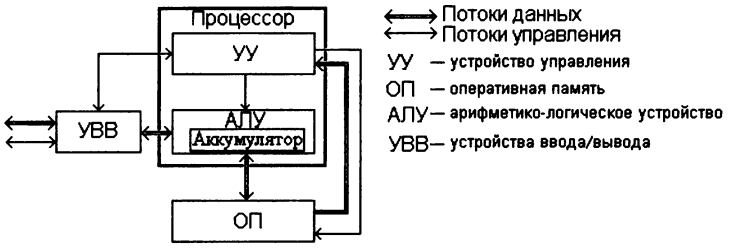


Рис. 5.4. Архитектура машины фон Неймана

Устройство управления процессора передает последовательности управляющих сигналов на все остальные устройства компьютера.

ВНИМАНИЕ

Последовательность обрабатываемых процессором кодов данных, выполняемых им команд и управляющих сигналов принято называть потоком данных, потоком команд и управляющим потоком соответственно.

В машине фон Неймана имеются потоки данных между оперативной памятью и устройством управления, между оперативной памятью и арифметико-логическим устройством, а также между устройствами ввода/вывода и арифметико-логическим устройством. При этом обмен данными между устройствами ввода/вывода и оперативной памятью происходит через процессор. Поэтому на время обмена вычисления по программе прекращаются. В машине IAS во внутренних линиях, а также в линиях связи с внешними устройствами данные и команды передаются в параллельном режиме.

Основу арифметико-логического устройства составляет единственный регистр, который называется аккумулятором. Типичная команда выбирает операнд из оперативной памяти и использует его в операции с аккумулятором, где при необходимости должен находиться второй операнд. Результат остается в аккумуляторе. Архитектура компьютеров, которая основана на этом принципе, называется **аккумуляторной архитектурой**.

Первой действующей машиной, в которой были реализованы базовые принципы фон-неймановской архитектуры, стал компьютер EDSAC (от Electronic Delay Storage Automatic Calculator — автоматический вычислитель с электронной памятью на линиях задержки), построенный М. Уилксом при участии Алана Тьюринга в Великобритании в 1949 г. В этой машине использовалась двоичная система счисления. Программа хранилась в оперативной памяти. Система команд была одноадресной. Тактовая частота машины составляла 0,5 МГц, то есть длительность такта равнялась 2 мкс. Но одна команда, занимая почти 5000 тактов, выполнялась в среднем за 0,01 с, что составляло примерно 100 арифметических операций за одну секунду. С машины EDSAC принято вести отсчет **первого поколения** компьютеров.

Развитие архитектуры первого поколения компьютеров происходило в основном в рамках фон-неймановской архитектуры. При этом наращивались количественные

показатели производительности компьютеров: увеличивались тактовая частота, длина машинного слова, скорость обмена у оперативной памяти. Повышалась плотность хранения информации, уменьшались размеры компьютеров. Появились новые долговременные носители программ и данных, такие как *магнитные ленты*, *барабаны* и *диски*, на которых научились хранить не только числовую, но и текстовую, звуковую, графическую информацию. Появились удобные средства для организации взаимодействия человека и машины, в том числе компактные и надежные клавиатуры, служащие для первичного ввода информации и управления работой компьютера, а также подобные телевизионным приемникам монохромные, а затем и цветные устройства для отображения информации — дисплеи. Так, например, первый дисплей с разрешением 512 × 512 пикселов появился в составе выпущенного в 1961 г. компьютера PDP-1.

В нашей стране первые компьютеры создавались примерно в тот же период. В 1947–1951 гг. под руководством академика С. А. Лебедева была пущена первая советская вычислительная машина — МЭСМ (малая электронно-счетная машина). Кроме того, выпускались машины «Стрела», «Минск», «Днепр», «Урал», БЭСМ (большая электронно-счетная машина), М-2, «Мир» и некоторые другие. Они разрабатывались под руководством крупных советских конструкторов и теоретиков И. С. Брука, М. А. Карцева, Б. И. Рамеева, В. М. Глушкова, Ю. А. Базилевского. Советская вычислительная техника того периода относилась к лидирующим в европейских странах.

5.3. Параллелизм в архитектуре начального периода

Практически сразу же после появления компьютеров в их архитектуру стали внедрять параллелизм, затрагивавший в первый период развития в основном функционирование автономного компьютера.

5.3.1. Параллельная обработка разрядов кода

Оперативная память у машин EDSAC и EDVAC реализована *на ртутных линиях задержки*, которые представляют собой тонкие герметичные металлические трубки с парами ртути внутри (рис. 5.5). На концах трубки находятся кристаллы кварца. По своим функциям они похожи на мембраны в телефонных аппаратах: один из кристаллов играет роль передающего, а другой — принимающего. К передающему кристаллу подсоединяется электрическое устройство, создающее колебания. Они, в свою очередь, возбуждают упругие колебания в парах ртути, которые с определенной временно́й задержкой доходят до другого конца трубки и заставляют колебаться приемный кристалл. Его колебания преобразуются в электрические импульсы и по цепочке обратной связи вновь подаются на передающий кристалл. Получается замкнутый контур, который в динамическом режиме сколько угодно долго сохраняет поступившие в трубку импульсы.

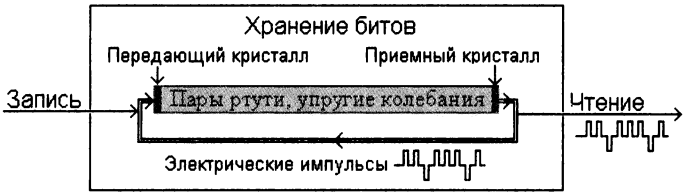


Рис. 5.5. Схема памяти на ртутных линиях задержки

Биты, которые нужно сохранить в памяти, подаются в виде электрических импульсов на передающий кристалл. Замкнутый контур памяти принимает их и сохраняет в описанном ранее динамическом режиме. При необходимости выполнить чтение информация снимается с линии обратной связи, как показано на рис. 5.5.

По сравнению с реализацией памяти в виде триггеров на электронных лампах, такая система отличается большей плотностью хранения данных. Например, трубка длиной 1 м может сохранять до тысячи бит. Если на этой же площади разместить ламповые триггеры, то в них можно сохранить только десятки бит. Кроме того, память на ртутных линиях задержки дешевле, чем ламповая память. Поэтому на ламповых триггерах были реализованы только самые ответственные узлы этих машин, в том числе их арифметико-логические устройства.

Однако память на ртутных линиях допускает только последовательную выборку записанных в нее битов. Следовательно, сложение, а также другие операции могут выполняться только поразрядно. Например, для сложения 16-битовых чисел требуется не менее 16 тактов работы процессора. Такая архитектура считается **разрядно-последовательной**.

В 1952 г. была пущена в эксплуатацию машина «Whirlwind 1», в которой оперативная память была построена на ферритовых магнитных сердечниках, представлявших собой колечки диаметром менее 1 мм. Материал, из которого они сделаны, обладает способностью длительное время сохранять одно из двух возможных состояний намагниченности (аналог северного и южного полюсов у магнита). Одно из этих состояний считается цифрой 0, а второе — цифрой 1 (рис. 5.6). Через каждое колечко проходит несколько проводов. Пропуская по ним ток нужных величины и направления, можно изменять состояние намагниченности или же определять текущее состояние, то есть осуществлять запись или чтение бита. Память на магнитных сердечниках оказалась гораздо компактнее, чем память на электронных лампах накаливания. Так, блок памяти объемом 1024 бита размещался на площади всего 12 × 12 см.

Отличительной особенностью памяти, построенной на ферритовых кольцах, является значительно более высокая скорость обмена, чем у памяти на ртутных линиях задержки. Машина «Whirlwind 1» установила рекордные по тому времени показатели быстродействия. Она выполняла в секунду 330 000 операций сложения и 60 000 операций умножения. С этого времени большинство компьютеров оснащались оперативной памятью на ферритовых кольцах. Ферритовая память была вытеснена только в конце 1960-х гг. памятью на транзисторах, а затем на интегральных схемах.

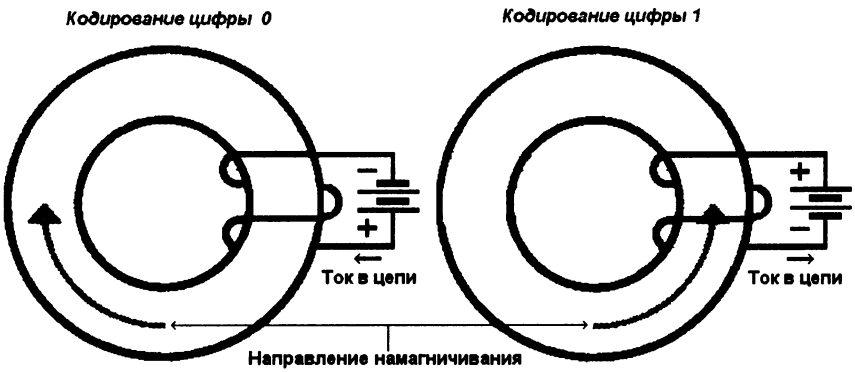


Рис. 5.6. Принцип сохранения двоичной цифры в ферритовом кольце

Не менее важным свойством памяти на ферритовых сердечниках является возможность одновременной, параллельной выборки всех битов машинного слова. Компьютеры, в которых организована параллельная выборка и обработка разрядов слова, относятся к **разрядно-параллельной архитектуре**. Впервые такой способ был реализован в выпущенном в 1953 г. компьютере IBM 701. При использовании параллельного способа обмена и обработки на сложение 16-битовых чисел требуется всего 1–2 такта работы процессора.

5.3.2. Совмещение во времени работы нескольких устройств

Ранее отмечалось, что в реализованной в машине IAS архитектуре фон Неймана обмен данными между внешними устройствами и оперативной памятью осуществлялся через процессор. Довольно быстро заметили, что такая организация компьютера весьма неэффективно использует самое дорогое его устройство — процессор. Он вынужден простаивать, пока очень медленно работающие внешние устройства завершат выполнение обмена.

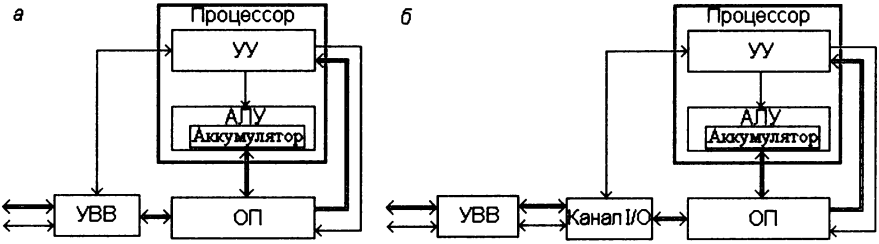


Рис. 5.7. Варианты архитектуры фон Неймана

На рис. 5.7, *а* представлен вариант архитектуры фон Неймана, в котором процессор непосредственно не участвует в выполнении обмена с внешними устройствами. В нем устройства ввода/вывода напрямую передают данные в оперативную

память, но процессор по-прежнему значительную часть времени контролирует процессы обмена. Выход нашли в совмещении во времени работы процессора и устройств, обеспечивающих обмен данными.

Канал ввода/вывода

Чтобы избавить процессор от обязанностей слежения за медленно работающими внешними устройствами, в состав машины включили независимые специализированные процессоры, которые могут выполнять только операции, связанные с обменом, и никаких других. Такие специализированные процессоры принято называть **каналами ввода/вывода**, или просто **каналами**. Впервые каналы ввода/вывода появились в составе выпущенной в 1958 г. машины IBM 709.

Именно из-за появления в составе компьютера дополнительных процессоров основной процессор стали называть **центральным**, а все остальные, дополнительные — **сопроцессорами**, **периферийными процессорами**, **процессорами ввода/вывода** и т.д.

Каналы выполняют обмен данными по специальным программам, которые по этой причине называются **канальными программами**. Как и обычные программы, канальные программы обмена могут находиться в оперативной памяти. В других вариантах архитектур канальные программы размещаются в собственной памяти канала.

В такой реализации архитектуры в обязанность процессора входит только запуск канала, а точнее, запуск канальной программы, реакция на завершение обмена и, возможно, на нестандартные ситуации в процессе обмена. В остальное время процессор может заниматься выполнением каких-либо других программ. Для реализации реакции процессора на завершение каналом операций по обмену или на возникновение отказа в работе устройства используется описанный ранее механизм прерываний (см. 4.2.7).

Очевидно, что для фактического получения выигрыша от совмещения во времени операций ввода/вывода и вычислительных действий центрального процессора в памяти компьютера на стадии выполнения должно находиться несколько программ. В то время когда одна из программ ожидает завершения операций по обмену, выполняемых для нее каналом, центральный процессор занят выполнением другой программы. После поступления сигнала прерывания о благополучном завершении обмена процессор может вернуться к выполнению ожидавшей программы.

ВНИМАНИЕ

Режим работы компьютера, при котором в оперативной памяти на стадии выполнения одновременно находится несколько программ, принято называть многопрограммным или мультипрограммным.

Параллелизм, осуществленный за счет независимого одновременного выполнения разными устройствами компьютера разных программ, не приводит к ускорению

выполнения отдельной программы. Но общая производительность вычислительной системы увеличивается, так как процессор успевает выполнить больше машинных команд за единицу времени. Следовательно, за один и тот же отрезок времени при наличии каналов будет выполнено больше программ, чем в исходном варианте архитектуры.

Вариант архитектуры фон Неймана с одним каналом изображен на рис. 5.7, *б*, где I/O — это сокращенное обозначение устройств ввода/вывода (от input/output). Отметим, что в состав компьютера могут входить несколько каналов, которые могут работать одновременно, выполняя обмен для одной и той же или для разных программ, при этом к одному каналу может быть подключено до 256 внешних устройств. Различают **селекторные** (от select — выбранный), **мультиплексные** (от multiplex — многократный) и **блок-мультиплексные** каналы. Селекторные каналы обслуживают высокоскоростные внешние устройства (диски, ленты). Один селекторный канал занят обслуживанием только одного из подключенных к нему внешних устройств до полного завершения обмена. Мультиплексные каналы обслуживают низкоскоростные внешние устройства (принтеры, графопостроители и т. д.). Такой канал обслуживает сразу несколько устройств, одновременно выполняющих обмен. Мультиплексный сигнал успевает по очереди обработать все поступившие со всех устройств байты данных, пока каждое из них считывает или записывает свои байты. Блок-мультиплексные каналы, в принципе, аналогичны мультиплексным, но обмен выполняется крупными блоками данных, а не байтами или машинными словами, как это делают мультиплексные каналы.

Контроллер устройства

Включение в состав компьютера каналов — устройств, которые могут работать независимо от центрального процессора и одновременно с ним, — повлекло за собой изменение общих представлений об организации функционирования компьютера. В исходной фон-неймановской архитектуре (см. рис. 5.4) предусматривается единый общий центр управления действиями всех устройств компьютера. В качестве такого центра выступает устройство управления центрального процессора. Все действия всех устройств компьютера привязываются к синхроимпульсам тактового генератора и имеют четко определенные моменты начала выполнения действия и его завершения. Такой режим работы называется **синхронным** (от гр. synchronos — одновременный).

Жесткая синхронизация совместного функционирования многочисленных устройств компьютера, которые работают с самыми разными скоростями, приводила к необходимости вставлять в работу процессора и других быстродействующих устройств *такты ожидания*, в течение которых они простаивали, ожидая, пока низкоскоростные устройства закончат свои действия. Понятно, что такая схема снижает общую эффективность работы компьютера.

Совершенно иным способом функционируют каналы ввода/вывода. Они начинают работу по специальному запускающему сигналу процессора и действуют самостоятельно, не привязываясь к синхроимпульсам тактового генератора.

Длительность выполнения действий может быть произвольной, а последовательность их выполнения регулируется не определенными моментами времени в последовательности тактовых импульсов, а началом или завершением определенного этапа в работе канала. Такой режим работы называется **асинхронным** (не синхронным).

Для повышения общей эффективности работы компьютера асинхронный режим стали активно использовать в работе и других устройств компьютера. Вместе с тем, в ситуациях, в которых требуется жесткая временна́я связь действий, попрежнему применяется синхронный режим работы.

Чтобы избавить процессор от обязанностей управления различными устройствами компьютера, а также при необходимости организовать асинхронный режим их работы практически для каждого устройства компьютера, были разработаны похожие на каналы собственные блоки управления, которые называются **контроллерами**.

ВНИМАНИЕ

Контроллером называется блок управления устройством компьютера, который обеспечивает его самостоятельное функционирование и при необходимости преобразование данных из используемого в устройстве формата к другому формату.

Контроллеры имеются не только у устройств, выполняющих операции ввода/вывода. Существуют контроллеры микросхем оперативной памяти, контроллеры внешних запоминающих устройств, таких как магнитные и оптические диски, контроллеры шины и т. д.

Фактически устройства компьютера состоят из двух частей: собственно устройства, выполняющего специфические для него функции, и контроллера — блока управления устройством, который содержит также большую часть электронных схем, обеспечивающих связь между управляемым устройством и другими устройствами компьютера. Для различных типов устройств разрабатываются различные контроллеры. Более того, для одного и того же типа устройств могут быть разработаны различные контроллеры.

Контроллер может быть конструктивно включен в состав устройства, которым он управляет, в этом случае его называют *встроенным*. Кроме того, контроллер может быть оформлен в виде отдельной микросхемы.

Отличительной особенностью контроллеров является их активный характер, проявляющийся в управлении устройством по некоторому подобию программ для центрального процессора или канала. В ряде контроллеров такие программы могут настраиваться с целью выбора более подходящего режима работы устройства. Для этого в состав контроллера включается независимый специализированный процессор, который выполняет свои функции по собственным программам. Контроллеры, обладающие такими возможностями, называются *программируемыми*.

Кроме контроллеров для подключения некоторых устройств используются и более простые блоки, которые называются **адаптерами**.

ВНИМАНИЕ

Адаптером называется блок соединения, сопряжения нескольких устройств компьютера с разными форматами представления данных.

В отличие от контроллера, адаптер не играет активной роли в работе устройства. Он не управляет его работой, а лишь пассивно преобразует данные из одного формата в другой. Обычно через адаптер к шине подсоединяются довольно простые внешние устройства персонального компьютера.

Буфер

Для организации асинхронного режима работы кроме контроллера необходимо еще одно устройство. Дело в том, что устройство, принимающее информацию (обычно это оперативная память), в момент завершения асинхронной передачи данных от одного из устройств может быть уже занято выполнением действий для другого устройства, и в этой ситуации оно не может принять новые данные. А передающее устройство во многих случаях не может повторить их передачу. Следовательно, переданные данные могут быть потеряны. Чтобы в подобных случаях не потерять данные, в конструкции компьютеров предусматриваются вспомогательные запоминающие устройства, которые называются **буферами**.

ВНИМАНИЕ

Буфером называется запоминающее устройство, помещаемое между двумя другими устройствами для организации асинхронной работы или для сглаживания разницы в скоростях обмена у этих устройств.

Если в момент передачи принимающее устройство занято и не может принять данные, то они направляются контроллером передающего устройства в буфер, где данные накапливаются и ожидают освобождения принимающего устройства (рис. 5.8, *а*). Как только приемное устройство оказывается свободным, контроллер организует передачу данных из буфера в это устройство.

Рис. 5.8. Схема использования: *а* — буфера; *б* — буферных областей памяти

Буфер, так же как и контроллер, может быть встроенным в устройство или выполненным в виде отдельной микросхемы. Объем буфера существенно зависит от обслуживаемого им устройства. Для одних устройств он может быть равным машинному слову, а для таких, например, как дисплей, доходить до десятков мегабайт.

Если буфер находится между устройствами, которые выполняют обмен с существенно разными скоростями, то временное накопление данных в буфере играет сглаживающую роль, обеспечивая высокоскоростному устройству более выгодный режим работы. Во время накопления в буфере данных, получаемых от низкоскоростного устройства, высокоскоростное устройство работает в холостом режиме. Когда в буфере сформируется достаточная для передачи порция данных, они передаются в скоростном режиме, удобном для принимающего устройства. Аналогичным образом выполняется и передача данных от высокоскоростного устройства к низкоскоростному. Вначале данные с высокой скоростью передаются в буфер, а затем с низкой — из буфера. Таким образом, каждое из участвующих в обмене устройств работает с наиболее подходящей для него скоростью.

В том случае, когда принимающим устройством является оперативная память, а не отдельный буфер, может быть организована **буферная область** памяти.

ВНИМАНИЕ

Буферной областью называется рабочая область оперативной памяти, предназначенная для временного хранения данных в процессе обмена. Она обеспечивает асинхронный режим работы и сглаживание разницы в скоростях обмена между внешним устройством и оперативной памятью. Накопление данных в буферной области для временного хранения в процессе обмена называется буферизацией.

Схема использования буферной области памяти приведена на рис. 5.8, *б*. Как правило, в буферной области образуется несколько буферов. В процессе заполнения одного из буферов данными от внешнего устройства программа с нужной ей скоростью выбирает данные из другого буфера. На рис. 5.8, *б* внешнее устройство заполняет буфер А, программа в это время выбирает данные из буфера В. После того как программа закончит выборку из буфера В, буфера поменяются ролями. В общем случае количество буферов и их объем могут регулироваться специальными настройками операционной системы. Если в области образовано несколько буферов, то они участвуют в обмене в циклическом режиме.

Распространяя идею совмещения во времени работы различных устройств, приходим к выводу о целесообразности организации одновременного обмена с оперативной памятью сразу для нескольких внешних устройств. При этом каждое из устройств должно иметь в оперативной памяти отдельную буферную область. Кроме того, возникает необходимость в организации **многовходовой (многоканальной)** памяти, которая допускает одновременное и независимое обращение к памяти со стороны нескольких устройств. Точнее, эта функция относится к контроллеру, обеспечивающему соответствующий режим работы. В персональных компьютерах IBM PC эти функции выполняет **контроллер DMA** (от Direct Memory Access — прямой доступ к памяти), который обеспечивает операции обмена внешних устройств с оперативной памятью по нескольким независимым каналам. Фактически

контроллеры DMA являются сопроцессорами ввода/вывода, освобождающими центральный процессор от длительных операций по обмену данными.

Существует два варианта начала операций по обмену данными в компьютере. В одном случае, согласно описанной ранее схеме работы каналов, обмен инициируется, запускается процессором, а во втором инициатором обмена выступают внешние устройства. Дальнейшее управление обменом в обоих случаях осуществляется контроллером DMA автономно от инициатора обмена. Далеко не все внешние устройства могут инициировать обмен с памятью. Те устройства, которые без вмешательства процессора могут запускать обмен по каналам DMA, входят в группу **Bus Mastering-устройств** (bus mastering — управление, владение шиной). К устройствам этой группы обычно относятся жесткие диски. Bus Mastering-устройство может иметь собственный контроллер, подобный контроллеру DMA. Многовходовая оперативная память, очевидно, обеспечивает более эффективное использование возможностей процессора и компьютера в целом.

5.3.3. Направления дальнейшего развития параллелизма

По мере накопления опыта конструирования и эксплуатации вычислительных систем фон-неймановской архитектуры специалисты осознавали ее недостатки и искали пути преодоления присущих ей ограничений. Выдающийся ученый и конструктор вычислительной техники академик С. А. Лебедев в докладе на сессии Академии наук СССР в 1957 г. сформулировал принципиальные положения, во многом предвосхитившие дальнейшие направления развития архитектуры вычислительных систем. Вот некоторые идеи Лебедева по развитию параллелизма в ЭВМ, представленные в виде выдержек из его доклада.

- «Нужно отказаться от стандартной схемы выполнения команды, когда выборка очередной команды заканчивается только после отсылки результата в оперативную память. Выполнение арифметических операций в значительной мере может быть совмещено во времени с обращениями в оперативную память. Нужно выполнять текущую команду и параллельно выбирать следующую, осуществляя опережающий просмотр команд. Вызов операндов очередной команды можно совместить во времени с отсылкой на хранение результатов выполнения предыдущей команды».
- «Вычисления можно ускорить, если организовать выборку команд и данных по нескольким направлениям, трактам, каналам, линиям связи».
- «Для уменьшения влияния самого существенного фактора — времени обращения к оперативной памяти, следует включить в состав ЭВМ *сверхбыструю* (курсив мой. — *А. Н.*) память относительно небольшого объема, которая должна стать промежуточным звеном между процессором и оперативной памятью. Заполнение этой памяти из оперативной должно производиться одновременно с вычислениями».
- «Следует включить в состав ЭВМ несколько одновременно работающих арифметико-логических устройств, каждое из которых имеет собственную сверх-

быструю память и собственное устройство управления, а также общую для всех оперативную память и устройство управления для всей машины в целом».

- «Нужно организовать параллельную работу нескольких процессоров в одной машине, а также нескольких машин, объединенных общим устройством управления и линиями связи для обмена данными».

Эти идеи в том или ином виде реализовывались в архитектуре разрабатываемых в разных странах мира компьютеров. В следующих разделах учебника обсуждаются подробности реализации предложений Лебедева в дальнейших разработках вычислительных систем.

Дополнительный материал по обсуждавшимся в главе вопросам можно найти в изданиях [2], [5], [20], [30].

Контрольные вопросы и упражнения

1. Обоснуйте тезис о необходимости широкого использования параллелизма в компьютерных системах.
2. Чем отличается собственно параллелизм от конвейеризации?
3. Охарактеризуйте отличительные особенности механического этапа в развитии вычислительных систем.
4. Опишите возможности и характерные особенности машин Бэббиджа.
5. Охарактеризуйте отличительные особенности электромеханического этапа в развитии вычислительных систем.
6. Охарактеризуйте отличительные особенности начального периода электронного этапа в развитии вычислительных систем.
7. Изложите базовые принципы организации вычислительных систем, выдвинутые фон Нейманом.
8. Изобразите структурную схему машины фон Неймана.
9. Опишите принцип действия памяти на ртутных линиях задержки.
10. Опишите принцип действия памяти на ферритовых кольцах.
11. Чем отличается разрядно-последовательная обработка кодов от разрядно-параллельной?
12. Какую роль играют каналы в работе компьютеров? Какие типы каналов вам известны?
13. Для чего нужны контроллеры? Какие бывают контроллеры? Чем отличаются контроллеры от адаптеров?
14. Опишите принцип работы контроллеров DMA.
15. В чем отличие контроллеров от каналов?
16. Чем отличаются контроллеры от адаптеров?
17. Для чего нужен буфер?
18. Сформулируйте изложенные академиком С. А. Лебедевым идеи по развитию архитектуры вычислительных систем.

Глава 6
Многопрограммный режим работы компьютеров

В упомянутом в 4.2.8 реальном режиме все ресурсы компьютера предоставляются в монопольное использование единственной выполняющейся программе — таким образом, компьютер работает в однопрограммном режиме. Используемый в этом режиме механизм сегментации памяти обеспечивает программе возможность доступа к любым полям оперативной памяти. Если сохранить эту возможность в многопрограммном режиме, который необходимо организовать для получения реального выигрыша от совмещения во времени операций ввода/вывода и вычислительных действий центрального процессора, то любая программа может по ошибке или целенаправленно записать некоторый код в поле памяти, принадлежащее другой выполняющейся в это же время программе. Тем самым будет уничтожен уже находящийся в этом поле код. В общем случае это приведет к искажению всей программы. Следовательно, одновременно выполняющиеся программы нуждаются в защите от взаимного влияния, от взаимного разрушения данных и программного кода. Именно поэтому многопрограммный режим работы процессоров Intel называется **защищенным**.

ВНИМАНИЕ

Реальный режим процессоров Intel однопрограммный. В этом режиме все модели процессоров Intel являются логическими аналогами процессора i8086. Защищенный режим процессоров Intel — многопрограммный. В этом режиме реализован комплекс мер по защите программ от взаимного влияния и разрушения.

В отличие от реального режима, в защищенном режиме аппаратные и программные ресурсы компьютера должны быть распределены неким образом между всеми одновременно выполняющимися программами. Общие ресурсы компьютера могут использоваться такими программами одновременно либо по очереди.

Вместе с тем любой программе для ее полноценного и безошибочного выполнения требуются и принадлежащие только ей ресурсы. Наиболее важным из таких ресурсов являются индивидуальные участки оперативной памяти, доступ к которым может получить только программа-хозяйка и не могут другие программы. Так же монопольно должны закрепляться за программой специальные таблицы, в которых описываются выделенные ей участки памяти. Поэтому в многопрограммном режиме вводится новое, более широкое по сравнению с понятием программы понятие **задачи**.

ВНИМАНИЕ

Задачей называется совокупность из выполняющейся программы и выделенных ей для выполнения аппаратных и программных ресурсов. Задача является основной единицей работы вычислительной системы в многопрограммном режиме. Адресным пространством задачи называется совокупность адресов оперативной памяти, по которым может обращаться выполняющаяся программа.

Для организации многопрограммной работы в защищенном режиме работы процессоров Intel предусмотрено несколько механизмов, обеспечивающих различные аспекты защиты. Это механизмы сегментации, виртуальной памяти, привилегированного доступа, прерываний и переключения задач.

Механизм *сегментации* оперативной памяти в защищенном режиме обеспечивает программам такой доступ к сегментам, который делает невозможным взаимное разрушение программ во время их одновременного выполнения. Сегментация оперативной памяти в защищенном режиме существенно отличается от сегментации в реальном режиме.

В многопрограммном режиме каждой программе выделяется несколько сегментов памяти. Современным программам требуется довольно много памяти, поэтому количество сегментов, выделенных программе, может быть довольно большим. А если в оперативной памяти находится не одна программа, а несколько, то имеющегося в компьютере фактического объема памяти может оказаться недостаточно. Следовательно, требуется механизм, который обеспечивает все выполняющиеся программы достаточным объемом оперативной памяти вне зависимости от реально имеющегося в компьютере ее объема. Таким механизмом является **виртуальная** память.

ВНИМАНИЕ

Виртуальной[1] памятью называется механизм, обеспечивающий выполняющимся программам доступ к оперативной памяти, объем которой ограничен только адресным пространством компьютера и не зависит от имеющегося в нем фактического объема оперативной памяти.

[1] Виртуальный — от лат. virtualis — воображаемый, возможный, такой, какой мог бы быть, может или должен быть.

В защищенном режиме сегментами оперативной памяти и виртуальной памятью управляют аппаратные средства компьютера, которые обеспечивают:

- компактность указания адреса в машинных командах программы;
- гибкость механизмов адресации, поддерживающих эффективную работу программы с данными любой сложной структуры;
- защиту адресных пространств задач;
- поддержку виртуальной памяти с возможным объемом до 4 Тбайт.

Механизм сегментации защищенного режима обеспечивает программам пользователей защиту от взаимного влияния, делая невозможным для любой такой программы доступ к «чужому» участку оперативной памяти. Вместе с тем операционная система для выполнения своих функций должна иметь доступ к любым участкам оперативной памяти, чтобы, например, при появлении ошибки в программе определить ее характер и местоположение. Таким образом, доступ к участку памяти должен зависеть от того, какая программа к нему обращается. Такое разграничение осуществляет механизм **привилегированного доступа**.

В многопрограммном режиме, так же как и в однопрограммном, во время выполнения программы могут происходить прерывания различных типов. Но в многопрограммном режиме обработка прерываний осложняется тем, что при наличии в системе нескольких центральных процессоров они могут возникнуть одновременно у нескольких программ. Поэтому механизм прерываний в защищенном режиме устроен сложнее, чем в процессоре i8086.

Неоднократно использованное ранее в качестве основного признака многопрограммного режима выражение «одновременно выполняющиеся программы» не совсем точно. Оно является более кратким заменителем выражения «программы, одновременно находящиеся на стадии выполнения». В многопрограммном режиме в оперативной памяти действительно может находиться несколько программ на стадии выполнения. Но если центральный процессор в системе только один, то он выполняет такие программы поочередно, выделяя каждой из них некоторый временной интервал, который принято назвать **квантом времени**. В силу того что процессор работает очень быстро, в *режиме квантования времени* во многих случаях создается иллюзия того, что программы выполняются одновременно. Для реализации поочередного выполнения программ необходим некоторый механизм, регулирующий переход процессора от выполнения одной программы к выполнению другой. Такое регулирование осуществляется механизмом **переключения задач**.

В следующих двух разделах рассматриваются важные для разработки прикладных программ вопросы организации памяти. Обсуждение механизмов привилегированного доступа, обработки прерываний и переключения задач выходит за рамки данного учебника. Подробное описание этих механизмов защищенного режима можно найти в [13], [25].

6.1. Сегментная модель памяти защищенного режима

Длина любого сегмента оперативной памяти в процессоре i8086 и в реальном режиме старших моделей процессоров всегда равна 64 Кбайт. Таким образом, сегмент обладает всего двумя атрибутами, *адресом* и *типом*. Различают три типа сегментов: сегмент кода, сегмент данных и сегмент стека. В зависимости от типа сегмента его адрес (адрес начального байта сегмента) записывается в соответствующий сегментный регистр. Программа может обращаться к любому полю в любом сегменте как для чтения, так и для записи.

Как отмечалось ранее, такой механизм сегментации, удобный для однопрограммного режима, в многопрограммном режиме не обеспечивает защиту программ от взаимного уничтожения. В связи с этим свойства сегментов памяти в защищенном режиме были изменены. Любые сегменты памяти в защищенном режиме имеют следующие атрибуты:

- адрес начала сегмента;
- длина сегмента;
- тип сегмента, определяющий способ его использования в программе;
- уровень привилегий, определяющий права данного сегмента относительно других сегментов.

6.1.1. Структура дескриптора сегмента

Значения или коды атрибутов сегмента оперативной памяти помещаются в специальное восьмибайтовое поле памяти, которое называется **дескриптором** (от describe — описывать) сегмента. Структура дескриптора сегмента памяти приведена на рис. 6.1.

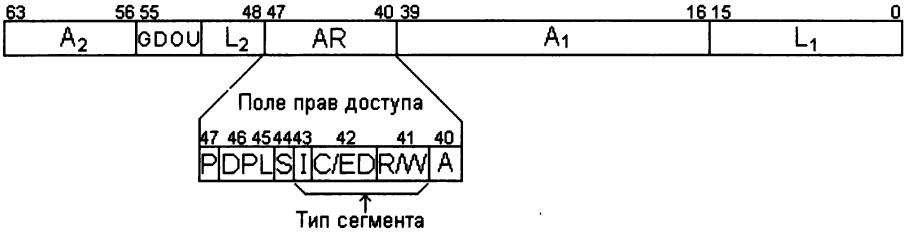


Рис. 6.1. Структура дескриптора сегмента

Единственный атрибут сегмента, смысл которого не изменился по сравнению с реальным режимом, — адрес сегмента. Однако этот адрес теперь не заносится в сегментный регистр процессора, а помещается в дескриптор сегмента, находящийся в оперативной памяти. Адрес сегмента занимает участки A_1 и A_2 дескриптора.

Их суммарная длина равна 32 разрядам, что соответствует адресному пространству объемом 4 Гбайт.

В отличие от реального режима, сегменты защищенного режима могут иметь любую длину от одного байта до 4 Гбайт. Длина сегмента, так же как и его адрес, записывается в дескриптор. Включение информации о фактическом размере сегмента в дескриптор обеспечивает возможность аппаратно контролировать работу программы с памятью, предотвращать ее обращение к несуществующим адресам либо по адресам, находящимся вне сегмента. Значение длины сегмента занимает участки L_1 и L_2 дескриптора, суммарная длина которых равна 20 бит. Длина сегмента может измеряться в байтах или страницах, длина которых всегда равна 4 Кбайт. Выбор единицы измерения длины сегмента зависит от значения бита **гранулярности G** (от лат. granulum — зернышко). При значении G = 0 длина сегмента измеряется в байтах, при этом 20-битовое поле длиной L (L_1 + L_2) обеспечивает возможность задания длины сегмента в пределах от 1 байта до 2^{20} байт = 1 Мбайт. При значении G = 1 длина сегмента измеряется в страницах. В этом случае она может изменяться в пределах от одной страницы (4 Кбайт) до 2^{20} страниц ($2^{20} \cdot 4$ Кбайт = 4 Гбайт).

В связи с необходимостью поддерживать совместимость с предшествующими 16-битовыми моделями в дескрипторе сегмента предусмотрен бит **разрядности D** (от Dimension — измерение), значение которого определяет используемую в сегменте разрядность операндов и адресов. Если значение D = 0, то используется 16-битовая, а при D = 1 — 32-битовая разрядность адресов и операндов.

В отличие от реального режима, в защищенном режиме возможны только два принципиально разных типа сегментов: сегмент кода и сегмент данных. Тип сегмента определяется значением бита **назначения сегмента I** (от Intending — предназначение). Если значение I = 0, то дескриптор описывает сегмент данных; если же I = 1, то это сегмент кода.

Сегмент стека считается самостоятельной разновидностью сегмента данных с особым способом изменения его длины. Стек растет в направлении уменьшения адресов оперативной памяти, в то время как у обычной разновидности сегмента данных рост происходит в направлении увеличения адресов памяти. В связи с этим разновидность сегмента данных уточняется с помощью бита **направления расширения ED** (от Expand Down — расширение вниз). Если значение ED = 0, то сегмент расширяется вниз, то есть в направлении увеличения адресов памяти (обычный сегмент данных). В противном случае, при ED = 1, сегмент растет в направлении убывания адресов памяти (стек). В дальнейшем изложении вместо оборота «сегмент данных с расширением в направлении убывания адресов» используется устоявшееся простое название «сегмент стека».

Любая выполняемая задача может создать и запустить на выполнение некоторую подчиненную задачу. В связи с этим различают обычные и подчиненные сегменты кода. Обычный сегмент кода содержит программу порождающей задачи, а подчиненный сегмент кода — программу порожденной, подчиненной задачи. Разновидность сегмента кода определяется значением **подчиненного** бита **C**

(от Conforming — подчиненный). Если значение C = 0, то сегмент кода считается подчиненным, а при C = 1 сегмент считается обычным. На самом деле биты ED и C — это не два разных бита в дескрипторе, а один и тот же бит, который принято обозначать **C/ED** и трактовать его смысл и значение в зависимости от значения бита I.

Программа, занимающая сегмент кода, всегда может быть выполнена. Запись в сегмент кода запрещена, а возможность чтения программного кода из этого сегмента, например с целью его копирования, регулируется битом **чтения/записи R/W** (от Readable/Writeable — читаемый/записываемый). Значение R/W = 0 запрещает чтение, а R/W = 1 — разрешает.

Чтение из сегмента данных разрешено всегда. Возможность записи в этот сегмент регулируется этим же битом R/W. Значение R/W = 0 запрещает запись, а значение R/W = 1 разрешает ее.

Таким образом, тройка битов I, C/ED и R/W определяет тип сегмента и допустимые операции с его содержимым. Все возможные комбинации этих битов и их смысл сведены в табл. 6.1. Отметим, что комбинация битов I = 0, C/ED = 1 и R/W = 0 определяет сегмент стека с запрещением в него записи. Организация такого стека, очевидно, смысла не имеет, поэтому код типа сегмента 010_2 считается не определенным.

Таблица 6.1. Типы сегментов памяти в защищенном режиме

I	C/ED	R/W	Код типа	Назначение сегмента
0	0	0	000	Сегмент данных, запись запрещена
0	0	1	001	Сегмент данных, запись разрешена
0	1	0	010	Не определено
0	1	1	011	Сегмент стека, запись разрешена
1	0	0	100	Подчиненный сегмент кода, чтение запрещено
1	0	1	101	Подчиненный сегмент кода, чтение разрешено
1	1	0	110	Обычный сегмент кода, чтение запрещено
1	1	1	111	Обычный сегмент кода, чтение разрешено

Код типа сегмента I, C/ED, R/W является составной частью байта **прав доступа AR** (от Access Right), входящего в дескриптор сегмента. В этот байт также входят (рис. 6.1) бит **присутствия P** (от Present), **код уровня привилегий дескриптора DPL** (от Descriptor Privilege Level), бит **система/сегмент S** (от System/Segment) и бит **доступа A** (от Accessed).

Бит присутствия P аппаратно получает значение 0, если в текущий момент времени сегмент отсутствует в оперативной памяти, и значение 1, если сегмент находится в ней. Бит доступа A также аппаратно получает значение 1 при обращении к сегменту (для чтения или записи), в противном случае его значение устанавливается в 0. Эти два бита используются механизмами виртуальной памяти.

Управляющая работой компьютера операционная система использует для выполнения своих функций различные объекты и структуры, которые так же, как и сегменты оперативной памяти, должны быть описаны. Удобно так организовывать работу системы, чтобы все эти описания имели единообразную структуру. В связи с этим разработчиками процессоров Intel было принято решение использовать для описания различных системных объектов те же самые дескрипторы, что и для описания сегментов памяти. Разумеется, внутренняя структура дескриптора в этом случае отличается от структуры дескриптора сегмента памяти. Для определения роли дескриптора служит бит система/сегмент S, который принимает значение 0, если дескриптор служит для описания системного объекта, и значение 1, если дескриптор используется для описания сегмента памяти.

В защищенном режиме любая программа не может обращаться к любому сегменту памяти, она должна иметь соответствующие права. Чтобы разграничить возможности программ по доступу к сегментам, введено четыре уровня привилегий. Закрепленный за сегментом уровень привилегий отображается в дескрипторе с помощью двухбитового кода уровня привилегий DPL. Наивысший приоритет имеют сегменты нулевого уровня, у которых код DPL равен 00_2, а минимальный уровень имеют сегменты с кодом DPL, равным 11_2.

Выполняющаяся программа имеет приоритет, совпадающий с приоритетом сегмента памяти, в котором она находится. Когда программа обращается к какому-либо сегменту для чтения или записи, то ее приоритет сравнивается с уровнем приоритета сегмента. Если приоритет программы больше приоритета сегмента или равен ему, то обращение разрешается, в противном случае обращение блокируется. Например, программа, имеющая DPL = 01_2, может обращаться за данными к сегментам памяти с DPL = 01_2, 10_2 и 11_2. Программа с приоритетом DPL = 00_2 может обращаться к любым сегментам, а программа с DPL = 11_2 — только к сегментам с таким же приоритетом.

Использование различных уровней привилегий обеспечивает возможность организовать приоритетное выполнение важных задач. Кроме того, обеспечивается гибкость механизмов доступа к памяти, которая дает операционной системе возможность восстановления нормальной работы в случае неправильной работы программ пользователей.

Кроме рассмотренных полей и битов в дескрипторе имеется не используемый бит **O** и используемый программистом по своему усмотрению бит **пользователя U** (от User). Подробная побитовая структура дескриптора сегмента памяти приведена на рис. 6.1. Инициализация (то есть заполнение разрядов поля значениями) дескриптора сегмента памяти выполняется операционной системой или программой или совместно операционной системой и программой.

6.1.2. Линейный адрес

В защищенном режиме в оперативной памяти выделяются **глобальные** сегменты памяти, к которым имеют доступ все выполняющиеся программы. Кроме того, каждой из выполняющихся программ выделяются индивидуальные **локальные**

сегменты памяти, к которым имеет доступ только программа-владелец этих сегментов и никакие другие программы доступа не имеют.

Дескрипторы сегментов памяти в зависимости от их назначения группируются в различные **дескрипторные таблицы**. Дескрипторы глобальных сегментов находятся в глобальной дескрипторной таблице **GDT** (от Global Descriptor Table), а дескрипторы локальных сегментов находятся в локальных дескрипторных таблицах **LDT** (от Local Descriptor Table) программы. Подчеркнем, что глобальная таблица GDT всего одна, а локальных таблиц много, точнее, столько, сколько одновременно выполняется задач. Элементы таблиц (дескрипторы) выделяются с помощью их номеров — индексов (рис. 6.2 и 6.3), которые входят в так называемые **селекторы** (от select — выделять).

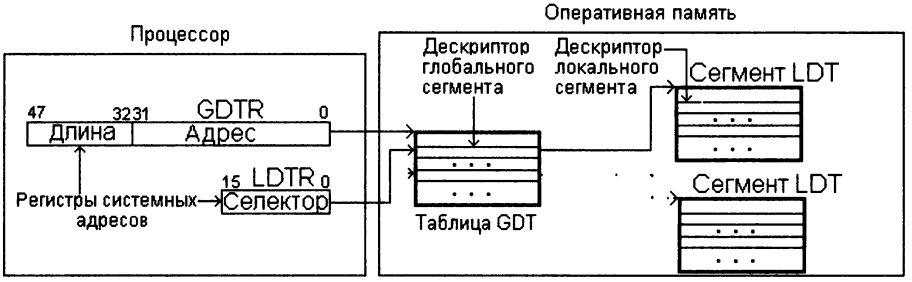


Рис. 6.2. Регистры системных адресов

Для определения местоположения таблиц GDT и LDT, а также некоторых других системных объектов применяются два различных способа, в которых используется содержимое регистров **системных адресов** процессора с длиной 6 и 2 байта.

Регистры системных адресов длиной 6 байт содержат 32-битовый адрес и 16-битовую длину таблиц или объектов, имеющих значение для всей системы. Адрес таблицы глобальных дескрипторов находится в системном регистре с названием **GDTR** (от Global Descriptor Table Register). Обращение по адресу, указанному в этом регистре, сразу приведет к начальному элементу дескрипторной таблицы (рис. 6.2).

Регистры системных адресов длиной 2 байта содержат селектор дескриптора сегмента памяти, в котором находится указываемый объект. С помощью селектора вначале нужно выбрать из таблицы GDT дескриптор соответствующего сегмента памяти, а потом из дескриптора выбрать адрес сегмента, в котором находится нужная таблица. Селектор дескриптора сегмента памяти, который содержит локальную дескрипторную таблицу, находится в системном регистре **LDTR** (от Local Descriptor Table Register).

Чтобы в защищенном режиме определить адрес какого-либо сегмента памяти, во-первых, необходимо знать, является этот сегмент глобальным или локальным, а во-вторых, нужен селектор дескриптора, выделяющий его из соответствующей таблицы. Эта информация находится в одном из сегментных регистров процессора, которые в защищенном режиме содержат не начальные адреса сегментов,

а селекторы дескрипторов. С их помощью из соответствующей таблицы GDT или LDT выбираются дескрипторы сегментов, содержащие не только начальный адрес, но и все остальные атрибуты сегмента.

Шестнадцатибитовый селектор содержит 13-битовый индекс дескриптора (рис. 6.4) и находящийся во втором разряде бит признака дескрипторной таблицы TI (от Table indicator). Значение бита TI = 0 означает, что дескриптор нужно выбирать из глобальной таблицы, а TI = 1 означает, что нужный дескриптор находится в локальной таблице.

Фактическая длина глобальной дескрипторной таблицы задается в 16-битовом поле длины в регистре GDTR. Отметим, что длина таблицы задается в байтах, следовательно, максимальная ее длина равна 2^{16} байт. Поскольку элементы таблицы имеют длину 8 байт, она может содержать не более 2^{13} дескрипторов. В то же время 13-битовый индекс селектора как раз и обеспечивает возможность адресовать $2^{13} = 8198$ дескрипторов в одной таблице. Таким образом, любая программа, в принципе, может использовать 8198 глобальных и 8198 локальных сегментов оперативной памяти.

Для определения адреса поля памяти в защищенном режиме, так же как и в реальном, нужно знать адрес сегмента и внутрисегментное смещение. Способ определения внутрисегментного смещения — эффективного адреса не отличается от используемого в реальном режиме. Для сегментов кода он выбирается из регистра `eip`, а для сегментов данных и стека формируется из содержимого индексных регистров `esi` и `edi`, базовых регистров `ebx`, `ebp` и смещения, заданного в команде `Dk` (рис. 6.3).

Более сложным способом определяется адрес сегмента. Операционная система автоматически формирует в регистре GDTR адрес глобальной дескрипторной таблицы. Эта таблица должна быть доступна любым выполняющимся программам, поэтому адрес в GDTR не изменяется при переключениях процессора на выполнение другой задачи. А вот в регистр LDTR операционная система при каждом переключении процессора на новую задачу загружает индивидуальный для каждой задачи селектор дескриптора из GDT, в котором находится адрес ее локальной таблицы LDT. Этот селектор доступен только программе-хозяйке. Таким образом, ни одна из программ пользователей не может получить доступ к локальной дескрипторной таблице другой программы.

Из заданного в команде сегментного регистра извлекается 16-битовый селектор. Из него выбирается бит признака дескрипторной таблицы TI и 13-битовый индекс. Бит признака TI используется для определения таблицы, в которой находится дескриптор, а индекс служит для выборки из соответствующей дескрипторной таблицы (GDT при TI = 0 и LDT при TI = 1) конкретного дескриптора сегмента памяти. Для определения адреса глобального сегмента нужно сделать следующее.

1. Прочитать из регистра GDTR адрес глобальной дескрипторной таблицы GDT.
2. Выбрать селектор из заданного в команде сегментного регистра.
3. Выбрать из селектора индекс дескриптора.

4. Определить адрес дескриптора сегмента, умножив индекс на 8 и сложив результат с адресом GDT.
5. Выбрать из найденного таким образом дескриптора искомый адрес глобального сегмента памяти.

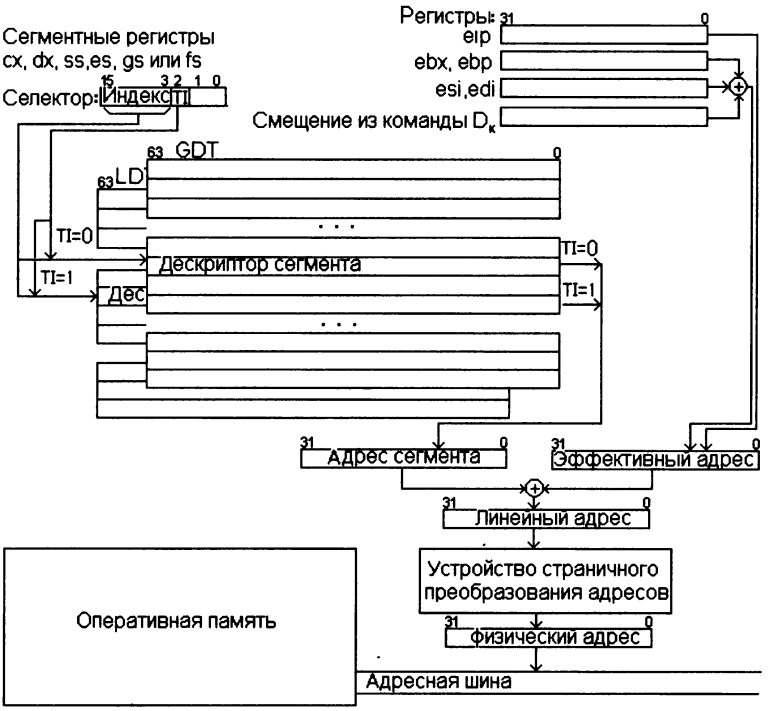


Рис. 6.3. Схема образования физического адреса в защищенном режиме

Для определения адреса локального сегмента вначале нужно описанным только что способом найти адрес сегмента, в котором находится таблица LDT, а затем уже в ней определить адрес локального сегмента.

1. Прочитать из регистра GDTR адрес глобальной дескрипторной таблицы GDT.
2. Прочитать из регистра LDTR селектор дескриптора сегмента памяти, который содержит таблицу LDT.
3. Выбрать из селектора индекс дескриптора в глобальной таблице.
4. Определить адрес дескриптора, умножив индекс на 8 и сложив результат с адресом GDT.
5. Выбрать из найденного дескриптора адрес сегмента памяти, в котором находится таблица LDT.
6. Выбрать селектор из заданного в команде сегментного регистра.
7. Выбрать из селектора индекс дескриптора искомого сегмента.

8. Определить адрес дескриптора, умножив его индекс на 8 и сложив результат с адресом таблицы LDT.
9. Выбрать из найденного дескриптора адрес искомого сегмента памяти.

Все эти выборки и преобразования осуществляются аппаратно с использованием специальных приемов, обеспечивающих высокую скорость получения адресов.

После определения 32-битового адреса сегмента он складывается с 32-битовым эффективным адресом. Полученный таким образом адрес называется **линейным**. Если механизмы виртуальной памяти отключены, то линейный адрес является одновременно *физическим*, или *исполнительным*, адресом, который выставляется на адресную шину и по которому происходит фактическое обращение в оперативную память (см. рис. 6.3). Определение физического адреса в режиме виртуальной памяти обсуждается в разделе 6.2.

В защищенном режиме перед обращением в память проверяется, не вышел ли определенный таким способом физический адрес за границы сегмента. Тем самым гарантируется защита одновременно выполняющихся программ от взаимного уничтожения. Кроме того, с помощью байта прав доступа определяются правомочность обращения программы к данному сегменту и возможность выполнения запрошенной операции. Если имеется какое-либо несоответствие, то обращение к сегменту блокируется и выдается соответствующее ситуации прерывание.

6.2. Организация виртуальной памяти

Для организации виртуальной памяти адресное пространство, то есть максимально возможный объем оперативной памяти, который для 32-битовой адресной шины состоит из 2^{32} байтов, подразделяется на 2^{20} = 1 048 576 блоков памяти размером 4 Кбайт каждый. Такие блоки принято называть **логическими страницами** (рис. 6.4).

Фактически имеющаяся в компьютере оперативная память также делится на участки длиной 4 Кбайт, которые принято называть **страничными кадрами**. Например, при фактическом объеме оперативной памяти 512 Мбайт она содержит 131 072 страничных кадра, что составляет 1/8 адресного пространства. Именно эта ситуация изображена на рис. 6.4.

В режиме виртуальной памяти часть логических страниц адресного пространства физически размещается в страничных кадрах оперативной памяти, а все не поместившиеся логические страницы записываются в специальный файл на магнитный диск. При этом программист считает все программы и данные находящимися в адресном пространстве процессора, то есть на логических страницах виртуальной памяти. Для него отсутствует разница между логическими страницами, фактически присутствующими в оперативной памяти, и станицами, находящимися на магнитном диске.

В распоряжении программиста находятся сегменты памяти, которые могут иметь любую длину. В то же время логические страницы имеют фиксированную длину,

поэтому одна страница может содержать несколько сегментов, один сегмент или какую-то его часть. Таким образом, сегмент может целиком входить в логическую страницу или занимать несколько страниц.

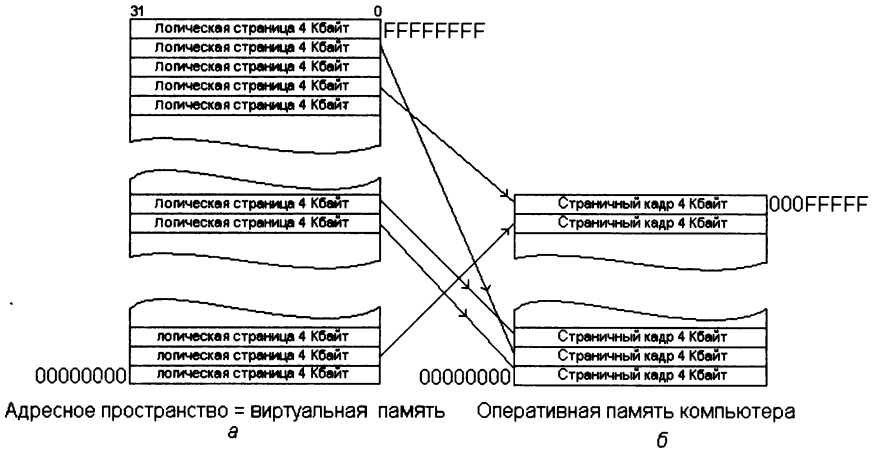


Рис. 6.4. Память: *а* — виртуальная; *б* — оперативная

При любом обращении к памяти с помощью бита P дескриптора сегмента определяется, находится ли связанная с сегментом логическая страница в оперативной памяти. Если страница находится в памяти, то происходит прямое обращение к ней. В противном случае нужная логическая страница автоматически загружается (**подкачивается**) операционной системой с жесткого диска в один из неиспользуемых или редко используемых страничных кадров оперативной памяти. Фиксированный размер всех страниц позволяет любую логическую страницу, фактически находящуюся на жестком диске, переписать в любой страничный кадр оперативной памяти. Для проверки использования страницы выполняющимися программами применяется бит A дескриптора сегмента, и для записи новой страницы выбираются только те кадры, в которых находятся сегменты со значением A = 0, то есть неиспользуемые сегменты. Старая логическая страница из выбранного страничного кадра переписывается на жесткий диск.

Поскольку на замену логических страниц требуется определенное время, выбор страничного кадра (из тех, у которых A = 0) для такой замены осуществляется по специальным алгоритмам, которые минимизируют необходимость ее выполнения.

Описанный процесс замещения логических страниц в страничных кадрах оперативной памяти называется **подкачкой,** или **свопингом** (от swap — замена). Для осуществления свопинга операционная система автоматически формирует на жестком диске файл, название которого зависит от операционной системы. Например, в операционной системе Windows он называется swap.sys или pagefile.sys. Файл подкачки и физическая оперативная память вместе взятые образуют виртуальную память, поэтому файл подкачки называется также *файлом виртуальной памяти.*

Ранее описан способ получения заданного в команде программы линейного адреса. Еще раз подчеркнем, что линейный адрес — это адрес в виртуальной памяти. Однако обращение к оперативной памяти может происходить только по физическим адресам. Поскольку в процессе подкачки любая логическая страница может попасть в любой страничный кадр, логическим адресам могут соответствовать различные физические адреса. Преобразование логических адресов в соответствующие им физические осуществляется аппаратным устройством **страничного преобразования адресов** (см. рис. 6.3). На его вход поступает линейный адрес, а на выходе формируется физический адрес и при необходимости замещается логическая страница в одном из страничных кадров. Адрес, сформированный устройством страничного преобразования, выставляется на адресную шину, откуда он считывается микросхемами памяти. В принципе, механизм виртуальной памяти может быть отключен, тогда физический адрес совпадает с линейным.

Дополнительный материал по обсуждавшимся в главе вопросам можно найти в изданиях [13], [25] и [33].

Контрольные вопросы и упражнения

1. Что дает использование многопрограммного режима?
2. Охарактеризуйте особенности реального и защищенного режимов работы микропроцессоров Intel.
3. Что такое задача? Для чего понадобилось введение этого понятия?
4. Опишите особенности механизма сегментации в защищенном режиме.
5. Перечислите атрибуты сегментов памяти в защищенном режиме.
6. Что представляет собой дескриптор сегмента? Опишите его общую структуру.
7. Какие типы сегментов памяти предусматриваются в защищенном режиме?
8. Опишите механизм уровня привилегий защищенного режима. Для чего понадобилось включение механизма привилегий?
9. Как формируется линейный адрес в защищенном режиме?
10. Что представляет собой виртуальная память? Почему возникла необходимость в ее реализации?
11. Опишите механизм реализации виртуальной памяти.
12. Как формируется физический адрес в защищенном режиме?

Глава 7
Повышение эффективности оперативной памяти

Для количественной оценки эффективности оперативной памяти применяется система характеристик, содержащая следующие параметры:

- тип микросхем памяти;
- объем памяти;
- допустимая тактовая частота системной шины;
- быстродействие микросхем памяти;
- скорость обмена или производительность;
- плотность хранения данных;
- отношение цены к объему — удельная стоимость, или стоимость хранения одного бита данных;
- надежность или достоверность хранения данных.

Тип микросхемы памяти определяется ее физическим внутренним устройством. От типа микросхемы существенно зависят быстродействие и производительность памяти, которые, в свою очередь, оказывают существенное влияние на общее быстродействие компьютера.

Микросхемы памяти принимают и выдают коды данных на системную шину, которая работает на определенной тактовой частоте. Поскольку все действия микросхем памяти и шины во время такой передачи должны быть строго согласованы, считается, что определяющим фактором для микросхемы памяти является *тактовая частота системной шины*, которая, таким образом, становится одной из важных характеристик оперативной памяти.

Быстродействие памяти определяется временем, затрачиваемым на выполнение операций записи данных в память и считывания из нее. Чем меньше это время,

тем быстрее и эффективнее память. Отметим, что у современных компьютеров параметры, связанные с быстродействием памяти, измеряются в долях секунды: миллисекундах (1 мс = 10^{-3} с), микросекундах (1 мкс = 10^{-6} с) и наносекундах (1 нс = 10^{-9} с). Наносекунда — настолько малый отрезок времени, что свет в вакууме за одну наносекунду проходит всего 29,98 см.

Кроме быстродействия микросхемы памяти характеризуются производительностью, скоростью обмена или пропускной способностью, которая равна объему передаваемых в память или из памяти данных в единицу времени. Скорость обмена измеряется в количестве передаваемых байт в секунду (байт/с) или в кратных единицах — килобайтах в секунду (Кбайт/с), мегабайтах в секунду (Мбайт/с) и гигабайтах в секунду (Гбайт/с). Не следует путать быстродействие и производительность — это связанные, но различные характеристики памяти. Более детально это различие обсуждается в разделе 7.2.

Плотность хранения информации измеряется количеством байт, которое может быть сохранено в единице измерения, характерной для устройства памяти: например, для жестких дисков — на единице площади запоминающей поверхности.

Удельная стоимость определяется как стоимость хранения одного байта (килобайта, мегабайта и т. д.). Обычно она измеряется в рублях (евро, долларах) на 1 Мбайт или 1 Гбайт памяти.

Достоверность хранения данных определяется вероятностью появления ошибки во время обмена или хранения данных в памяти.

В этом разделе рассматриваются характеристики некоторых типов микросхем оперативной памяти и основные приемы их улучшения.

7.1. Статическая и динамическая память

Реализация памяти, основанная на обсуждавшихся в 3.3 триггерах, состоящих не менее чем из шести транзисторов, отличается тем, что после записи значения в триггер его состояние может сохраняться сколь угодно долго без применения каких-либо специальных действий (кроме постоянного наличия электропитания). Поэтому такую разновидность памяти назвали **статической**. Ее принято обозначать аббревиатурой **SRAM** (от Static Random Access Memory — статическая память с произвольным доступом).

Статическая память отличается высоким быстродействием, но в то же время она характеризуется довольно низкой плотностью хранения данных — в одном корпусе микросхемы можно разместить около миллиона байт. При современных требованиях к объемам оперативной памяти в компьютере статическая память считается дорогой и громоздкой. Кроме того, такая память отличается высоким энергопотреблением.

В настоящее время биты оперативной памяти реализуют с помощью конденсаторов. Считается, что конденсатор с высоким уровнем напряжения между обкладками содержит цифру 1, а с низким — цифру 0. Для обслуживания конденсатора к нему подсоединяется всего один транзистор (сравните с шестью транзисторами триггера). Такая реализация бита имеет значительно более высокую плотность хранения данных — в одном корпусе микросхемы можно поместить десятки миллионов байт. Это значит, что при равных объемах такая память будет занимать гораздо меньшую площадь, чем статическая разновидность. Кроме того, эта память отличается гораздо меньшей стоимостью и более низким потреблением электроэнергии.

Недостатком оперативной памяти на конденсаторах являются ее относительно низкие скоростные показатели. Есть еще один негативный фактор — реальные конденсаторы через несколько миллисекунд самопроизвольно теряют заряд. Чтобы находящиеся в оперативной памяти данные и программы при этом сохранялись, состояние конденсаторов необходимо периодически восстанавливать до исходного уровня. Для этого осуществляются периодическое считывание хранящихся в памяти данных и повторная их запись. Такой процесс называется **регенерацией памяти**. Регенерацию битов памяти можно совместить с чтением или записью данных. Можно также выполнять ее отдельно, но во время самостоятельной регенерации оперативная память недоступна для взаимодействия с процессом и другими устройствами компьютера, что может отрицательно сказаться на скорости работы компьютера. Тем не менее самостоятельные циклы регенерации необходимо с определенной периодичностью выполнять, поскольку без них при отсутствии операций чтения/записи занесенные в память коды окажутся утерянными.

Оперативную память, реализованную в виде периодически подзаряжаемых конденсаторов, называют **динамической** и обозначают **DRAM** (от Dynamic Random Access Memory — динамическая память с произвольным доступом).

7.2. Микросхемы памяти

Конструктивно биты как статической, так и динамической памяти объединяют в микросхемы. Группу битов микросхемы, которые считываются или записываются одновременно, принято называть **ячейкой микросхемы**. Не следует путать ячейки микросхемы с ячейками (байтами) оперативной памяти, которые могут совпадать или не совпадать друг с другом. Количество битов в ячейке микросхемы, то есть ее разрядность, может равняться 2, 4, 8, 16 и т. д. Для обеспечения оптимальных геометрических размеров микросхемы ее ячейки образуют матричную структуру (рис. 7.1). Индексы строки и столбца (на рисунке не показаны), на пересечении которых находится ячейка, образуют ее адрес внутри микросхемы. Обычно применяют квадратные матрицы ячеек микросхемы, хотя существуют и прямоугольные.

Рис. 7.1. Упрощенная структура микросхемы оперативной памяти

Важной характеристикой микросхемы является **структура**, определяющая общий объем микросхемы и разрядность ее ячеек. Структура задается в виде произведения объема на разрядность. При этом первый сомножитель, объем, задают в килобайтах или мегабайтах, а второй, разрядность ячейки, измеряется в битах. Единицы измерения килобайт и мегабайт принято сокращать до первой буквы. В примере на рис. 7.1 объем микросхемы равен 32 байтам, а ее ячейка содержит 4 бита кода; следовательно, эта микросхема имеет структуру 0,03125К × 4. Современные микросхемы памяти имеют разрядность ячеек 8, 16 бит и более, а их объемы достигают десятков мегабайт.

Для повышения быстродействия и согласования с разрядностью шины данных отдельные микросхемы могут быть объединены в так называемые **банки памяти**. Например, две микросхемы с разрядностью ячейки 8 бит могут объединиться в банк для формирования 16-битового машинного слова. В последнее время в связи с ростом объемов и разрядностей с той же целью внутри одной микросхемы образуют несколько независимых банков.

ВНИМАНИЕ

В информатике банком называется совокупность совместно используемых однотипных элементов, средств или устройств.

Отличительной чертой микросхем или ячеек, принадлежащих одному и тому же банку памяти, является возможность участвовать в одной и той же операции чтения или записи для любой ячейки выбранного банка и одновременное отсутствие такой возможности для всех ячеек всех остальных банков.

Необходимо понимать, что объединение микросхем в банки или разбиение микросхемы на банки производится только на функциональном, логическом уровне. Конструктивно, физически микросхемы оперативной памяти располагаются на так называемых **платах**, которые представляют собой тонкие пластинки обычно прямоугольной формы стандартных унифицированных размеров (например, 60 × 15 мм). Материал, из которого изготавливается такая пластинка, является

изолятором электрического тока. Микросхемы памяти и процессоров крепятся к платам с помощью упоминавшихся ранее сокетов и слотов, к которым подводятся размещаемые прямо на поверхности плат шины компьютера. На платах выполняется монтаж основной части всех электронных схем компьютера, к ним же прикрепляются некоторые внутренние устройства компьютера. Кроме того, на платах имеются различные разъемы для их крепления к корпусу компьютера и друг к другу, а также для соединения с линиями передачи данных, управляющих сигналов и электропитания. Каждая плата компьютера имеет специальное функциональное назначение: например, плата управления звуком — звуковая плата, плата управления изображением — видеоплата, плата управления сетевыми соединениями — сетевая плата и т. д. Среди плат компьютера выделяется одна главная плата, которую называют **системной**, или **материнской** платой. Именно к ней крепятся микросхемы процессора, по ней проходят шины, на которых располагаются сокеты, слоты и порты.

В современных компьютерах микросхемы памяти не крепятся на материнскую плату, а объединяются на отдельных платах, которые принято называть *модулями* памяти (рис. 7.2). Платы модулей памяти с помощью *краевых разъемов* вставляются в слоты материнской платы, и из них в конечном счете набирается вся оперативная память компьютера. Например, память объемом 512 Мбайт может состоять из четырех модулей по 128 Мбайт, из двух модулей по 256 Мбайт или из одного модуля объемом 512 Мбайт. В свою очередь, каждый модуль объемом 256 Мбайт может состоять из 16 микросхем по 16 Мбайт или из 8 микросхем по 32 Мбайт. Могут быть реализованы и другие варианты.

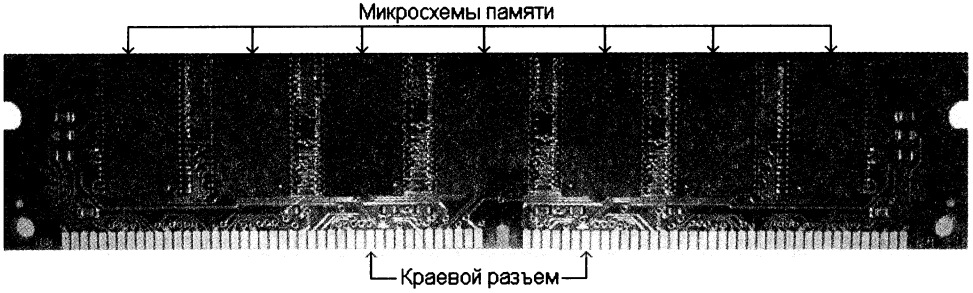


Рис. 7.2. Модуль памяти

Различают две конструкции модулей памяти: *SIMM* (от Single In Line Memory Modules — однорядные модули памяти) и *DIMM* (от Dual In Line Memory Modules — двухрядные модули памяти). В соответствии с названием в модулях SIMM микросхемы размещены в один ряд, а в модулях DIMM — в два ряда. Фактический объем оперативной памяти набирается из нескольких модулей того или иного типа. Одновременное использование разнотипных модулей памяти в современных компьютерах не предусмотрено. Поэтому при наращивании, расширении оперативной памяти за счет установки дополнительных модулей

приходится учитывать тип уже установленных. Следует отметить, что модули памяти типа SIMM считаются устаревшими, они постепенно выходят из употребления. Модули DIMM в настоящее время могут иметь объем 128, 256, 512 Мбайт и т. д. до 4 Гбайт. Количество контактов, или *пин* (от pin — штырь, вывод), у современных модулей памяти составляет 168, 184 и более.

Фактический диапазон адресов оперативной памяти распределяется между микросхемами модулей в соответствии с их объемами и количеством. Пусть, например, оперативная память состоит из четырех микросхем. Тогда старшие два бита адреса байта или поля памяти могут использоваться для выбора микросхемы или банка, к которым относится этот адрес (рис. 7.3). Например, адреса, начинающиеся с двух двоичных нулей, $00xxx...x_2$ (xxx — остальные цифры адреса), закрепляются за микросхемой с номером 0, адреса, имеющие вид $01xxx...x_2$, — за микросхемой с номером 1 и т. д. Последняя группа адресов, $11xxx...x_2$, закрепляется за микросхемой 3.

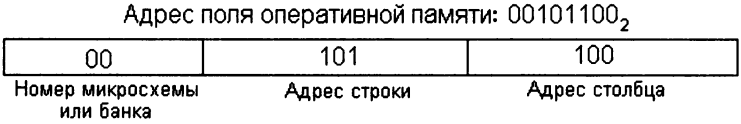


Рис. 7.3. Принцип выделения адреса ячейки микросхемы из адреса байта памяти

Как указывалось ранее, количество контактов у корпусов современных микросхем перевалило за сотню. Вместе с тем геометрические размеры корпусов в немалой степени зависят от количества контактов. Так, например, собственная площадь вентилей современной микросхемы составляет 200–300 мм^2, в то время как площадь корпуса может превышать 1000 мм^2. Эта разница обусловлена необходимостью пространственно разделять соседние контакты, чтобы поступающие на них сигналы не искажали друг друга. Поэтому разработчики интегральных схем стремятся уменьшить количество контактов на их корпусах.

Для уменьшения количества адресных контактов у микросхем памяти применяется следующая схема адресации ячеек. Оставшиеся после выделения номера микросхемы биты адреса (xxx...x из рассмотренных примеров) делятся на две группы. Биты старшей группы используются для выделения строки в матрице ячеек микросхемы, а биты младшей группы — для выделения столбца. Пусть, например, после выделения номера микросхемы в адресе остались цифры 101100_2. Тогда код 101_2 используется для выделения строки матрицы, а код 100_2 — для выделения ее столбца (см. рис. 7.3). Номера строки и столбца нужной ячейки матрицы передаются последовательно: сначала номер строки, а затем номер столбца, по одним и тем же адресным линиям микросхемы. Такой способ передачи адреса позволяет, например, при изображенных на рис. 7.2 трех адресных линиях с помощью декодера (см. 3.2.11) адресовать 8 строк и 8 столбцов, а всего 64 ячейки микросхемы.

7.3. Цикл памяти

Работа микросхем оперативной памяти во время чтения или записи кодов организована в виде циклически повторяемой последовательности действий.

ВНИМАНИЕ

Циклом памяти называется последовательность действий, которые выполняются в процессе чтения кода из микросхемы памяти или записи в нее нового кода.

Цикл памяти и циклы других устройств компьютера, таких как, например, шина, принято изображать в виде временны́х диаграмм. Поэтому вначале скажем несколько слов о традиционно принятом в литературе способе изображения таких диаграмм.

На временны́х диаграммах, одна из которых приведена на рис. 7.4, изображаются *графики изменения во времени* управляющих сигналов, значений на адресных линиях, на линиях шины данных и т. д. При этом нужно помнить, что каждый управляющий сигнал, адресный бит или бит данных в любой момент времени может принимать только одно из двух возможных значений, 0 или 1.

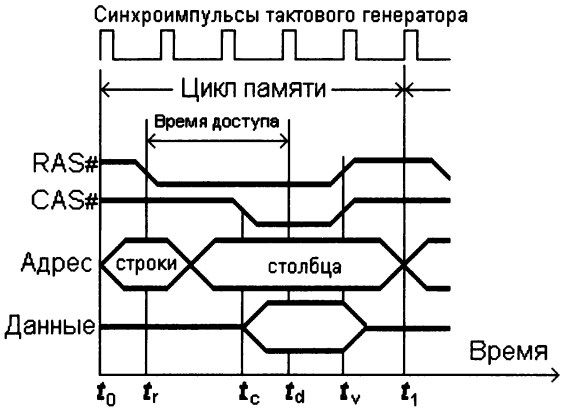


Рис. 7.4. Упрощенная временна́я диаграмма цикла памяти микросхемы

Стационарное (неизменное во времени) состояние управляющего сигнала (например, на рис. 7.4 такими сигналами являются RAS# и CAS#) или бита изображают в виде прямой горизонтальной линии. Если состояние сигнала с течением времени изменяется, то он изображается в виде нескольких отрезков горизонтальных линий, причем вышележащий отрезок соответствует значению 1, а нижележащий — значению 0. Во многих ситуациях временем изменения уровня сигнала от нулевого до единичного (или наоборот) пренебрегают и считают, что такой переход происходит скачком, мгновенно, как, например, показано на рис. 2.19 и 3.10.

Однако для рассматриваемых процессов с характерным временем всего в единицы наносекунд из-за конечности скорости света пренебрегать переходными

процессами уже невозможно. В связи с этим переход от одного уровня сигнала к другому изображается в виде связывающей эти уровни наклонной линии. Время от начала изменения сигнала (от начала наклонной линии) до его завершения (до конца наклонной линии) считается *переходным процессом*, а время от начала изменения сигнала до достижения примерно его среднего уровня считается временем *стабилизации сигнала*. До окончания стабилизации сигнала его использование для управления или других действий еще невозможно, потому что неясно, то ли это случайное искажение уровня напряжения, то ли целенаправленное его изменение.

Если по линиям шины данных или адресной шины одновременно передается несколько битов данных или адреса, то на временно́й диаграмме это отображается в виде двух параллельно идущих прямых. Переходный процесс для группы битов изображается в виде двух пересекающихся наклонных линий. На рис. 7.4 таким образом представлены диаграммы **Адрес** и **Данные**.

Теперь, не углубляясь в детали, рассмотрим функционирование микросхемы в течение цикла памяти, временна́я диаграмма которого изображена на рис. 7.4. Контроллер оперативной памяти получает от процессора запрос на выполнение чтения или записи, а также адрес байта или поля памяти, по которому нужно выполнить заданную операцию. При необходимости выполнить запись по шине данных поступает еще и записываемый код. Для определенности вначале рассмотрим цикл, связанный с чтением кода из ячейки микросхемы.

По полученному адресу байта или поля памяти контроллер определяет микросхему, в которой находится соответствующая этому адресу ячейка, и устанавливает для этой микросхемы (точнее, для банка) не изображенные на рис. 7.4 управляющие сигналы `CS` (от Chip Select — выбор чипа, микросхемы) и `RW` (от Read/Write — чтение/запись).

ПРИМЕЧАНИЕ

> Часто используемые в литературе обороты «устанавливает сигнал» или «выставляет сигнал» являются сокращением для выражения «формирует значение 1 на линии, по которой передается сигнал». Некоторые управляющие сигналы считаются установленными, если они получили значение 0. В этом случае применяется также оборот «активный низкий уровень сигнала». На временных диаграммах это отличие отражается добавлением знака «#» в название сигнала.

Микросхема (банк), получившая сигнал `CS`, становится *активной*, то есть способной к приему адреса и выполнению заданной сигналом `RW` операции чтения или записи. Затем на входы микросхемы подается адрес строки матрицы. Этот момент считается началом цикла памяти, на рис. 7.4 он обозначен t_0. После того как код на адресных линиях микросхемы стабилизируется, контроллер подает на микросхему сигнал `RAS#` (от Row Address Strobe — строб адреса строки). Обратите внимание на наличие символа «#» в названии сигнала, который означает, что активным является его нулевое, а не единичное значение.

Момент времени t_r, после которого сигнал RAS# считается стабилизировавшимся, рассматривается как момент появления в микросхеме запроса на чтение или запись. Получив нулевое значение сигнала RAS#, декодер микросхемы считает уже находящийся к этому времени на адресных линиях код адресом нужной строки микросхемы и делает всю строку активной, выставляя 1 на ее вход. Если выполняется операция чтения (RW = 0), то значения всех ячеек строки дублируются в специальный регистр микросхемы (на рисунке не показан), который играет роль своеобразного буферного устройства.

Затем, через достаточное для стабилизации считываемых из строки данных время контроллер выставляет на адресные линии адрес столбца микросхемы. Этот период на диаграмме **Адрес** изображен скрещивающимися наклонными линиями между участками **Строка** и **Столбец**, которые отображают постепенное исчезновение на адресных линиях одного кода и постепенное же появление другого. После стабилизации нового кода на адресных линиях контроллер выдает на микросхему сигнал **CAS#** (от Column Address Strobe — строб адреса столбца). Получив его, декодер трактует код на адресных линиях как адрес столбца и выделяет соответствующую ячейку в буферном регистре.

Момент стабилизации сигнала CAS# (обозначенный на временно́й диаграмме t_c) завершает в цикле памяти *фазу адреса* и начинает *фазу данных*. В этот момент код из выделенной ячейки появляется на связанных с шиной данных выходных линиях микросхемы. После стабилизации этого кода в некоторый момент времени t_d начинается его передача по шине. Выждав время, необходимое для приема шиной кода из ячейки, контроллер переводит сигналы *CAS# и RAS# в исходное состояние (момент* t_v), завершая этим фазу данных.

Однако в момент t_v микросхема еще не готова начать следующий цикл чтения или записи другого кода. Необходимо некоторое время на приведение микросхемы в состояние готовности к дальнейшей работе. Этот период в цикле памяти микросхемы считается *фазой восстановления*. Во время этой фазы восстанавливаются исходные уровни всех управляющих сигналов и, кроме того, происходит регенерация не только считанного из ячейки кода, но и кодов всех ячеек ее строки.

Момент t_1 окончания фазы восстановления считается моментом завершения текущего цикла памяти. В этот момент может быть начат следующий цикл чтения или записи.

Цикл записи в основном выполняется так же, как и цикл чтения, только после завершения фазы адреса данные с входных линий микросхемы передаются в выделенную ячейку микросхемы. Заметим, что фаза восстановления после цикла записи включает в себя также этап регенерации всех ячеек строки. Напомним, что для ячеек, не затронутых регенерацией, которая выполняется во время циклов чтения или записи, с определенным периодом выполняются специальные циклы регенерации.

Отрезок времени, который уходит на подготовку к чтению или записи кода, является очень важной характеристикой любых запоминающих устройств, в том числе и оперативной памяти. На диаграмме, изображенной на рис. 7.4, это интервал между точками t_r и t_d.

ВНИМАНИЕ

Время доступа памяти представляет собой задержку начала получения данных из памяти относительно появления запроса на них.

Отсчет времени доступа у микросхем памяти начинается от момента t_r стабилизации сигнала RAS#. Время доступа складывается из:

1) времени приема адреса строки;
2) времени замены на адресных линиях адреса строки на адрес столбца;
3) времени приема сигнала CAS#;
4) времени стабилизации выходных сигналов на шине данных во время чтения или времени стабилизации кода, переданного с входных линий данных.

Типичное время доступа для микросхем динамической памяти составляет 40–100 нс, а у лучших образцов достигает 5–10 нс. Время доступа у микросхем статической памяти исчисляется единицами наносекунд.

ВНИМАНИЕ

Длительность цикла памяти определяется как минимальный период следующих друг за другом обращений к памяти для выполнения чтения или записи.

Длительность цикла памяти микросхем включает в себя длительности всех его фаз. Как выяснится в дальнейшем, некоторые циклы памяти могут не содержать фазы адреса. Поэтому длительность цикла памяти может быть меньше времени доступа.

Упоминавшееся ранее быстродействие оперативной памяти определяется как временем доступа, так и длительностью цикла памяти, поскольку эти параметры описывают разные аспекты функционирования микросхемы. Чем меньше время доступа и длительность цикла памяти, тем микросхема быстрее и эффективнее, тем выше общее быстродействие компьютера.

Пропускная способность памяти зависит от длины машинного слова и от длительности цикла. Например, при длительности цикла 40 нс за одну секунду совершается 25 000 000 циклов памяти. Пусть разрядность шины данных равна 16 битам, то есть за один цикл передается 2 байта. Следовательно, за одну секунду передается 50 000 000 байтов. Таким образом, скорость обмена равна 50 000 000 байт/с, или примерно 48 Мбайт/с. Необходимо учитывать, что в современных типах микросхем данные выставляются на шину два или более раз за один цикл. Поэтому при расчете производительности микросхем нужно учитывать еще и этот фактор.

Обсуждаемый параметр — быстродействие микросхем оперативной памяти — характеризует максимальную (или **пиковую**) скорость обмена, которая достижима только на короткий период последовательной передачи из одного банка. При смене банка, с которым выполняется обмен, неизбежны существенные задержки и временны́е потери, следовательно, средняя производительность микросхем окажется ниже, чем рассчитанная на основании длительности цикла.

Во многих случаях использование для описания цикла памяти приведенной на рис. 7.4 временно́й диаграммы сложно и избыточно. Для упрощенного описания циклов памяти используются условные схемы, в которых длительность цикла задается в количестве тактов синхронизации центрального процессора. При этом в схеме обычно показывают длительности четырех следующих друг за другом циклов памяти.

В примере (см. рис. 7.4) в течение одного цикла памяти тактовый генератор выдает 5 синхроимпульсов. Следовательно, в стандартном режиме работы микросхемы такой цикл памяти изображается как условная схема вида 5–5–5–5, что соответствует четырем последовательным операциям обмена, выполненным за 5 тактовых импульсов каждая.

7.4. Типы микросхем динамической памяти

У рассмотренных стандартных микросхем DRAM цикл памяти включает в себя все три фазы: адреса, данных и восстановления. Поэтому они отличаются большими значениями времени доступа и длительности цикла памяти. Для повышения эффективности работы микросхем, уменьшения времени доступа и длительности цикла памяти были предложены различные варианты. Далее рассматриваются методы, сыгравшие значительную роль в улучшении характеристик оперативной памяти.

7.4.1. Расслоение памяти

Ранее упоминалось о том, что для повышения эффективности несколько микросхем памяти могут быть объединены в один банк, равно как в одной микросхеме может быть организовано несколько банков памяти. Напомним: банк отличается тем, что в текущей операции обмена могут участвовать только ячейки, относящиеся к активному банку, независимо от их принадлежности к одной и той же или к разным микросхемам.

С помощью соответствующих изменений в работе контроллера памяти соседние циклы чтения/записи для различных банков могут быть частично совмещены во времени. Это означает, что пока в одном из банков идет, например, фаза восстановления текущего цикла памяти, в другом банке памяти в это же самое время может проходить фаза адреса следующего цикла. Такой метод работы памяти принято называть *расслоением памяти*, *чередованием адресов* или *интерливингом* (от Interleave Mode — букв. режим прослаивания). Впервые режим расслоения памяти был реализован в компьютере STRETCH в 1961 г.

Метод расслоения памяти базируется на том, что, по практическим наблюдениям, адреса логически связанных величин программы чаще всего располагаются друг за другом в соседних байтах памяти. Следовательно, за счет совмещения во времени соседних циклов обращения в память можно получить определенный общий выигрыш во времени.

При отсутствии чередования банки активизируются сигналами CS последовательно, в зависимости от того, к какому из них поступил запрос. После выборки кода из текущего банка он переводится в неактивное состояние, и только затем активизируется следующий банк (рис. 7.5, *внизу*).

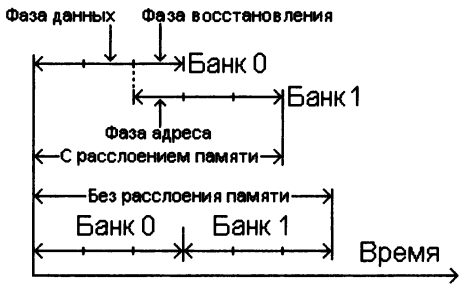


Рис. 7.5. Циклы с расслоением и без расслоения памяти

Теперь рассмотрим схему повышения быстродействия памяти для простейшего случая расслоения, когда используются два банка памяти. В этом случае адреса между банками распределяются так, чтобы в одном банке находились ячейки, соответствующие байтам с четными адресами, а во втором — с нечетными. Во время выборки, например, из банка с нечетными адресами в банке с четными адресами проходит фаза адреса для следующего выбираемого байта. То есть фаза восстановления одного банка накладывается на фазу адреса другого (рис. 7.5, *вверху*). Несмотря на то что длительность цикла для каждого банка в отдельности не изменилась, расслоение памяти уменьшает общее время выборки двух кодов и, следовательно, увеличивает общую производительность оперативной памяти.

В общем случае количество банков k выбирается равным небольшой степени двойки $k = 2m$, где $m = 1, 2, 3, 4$. При обращении к байтам памяти с последовательными адресами время выборки может уменьшиться в k раз.

Рис. 7.6. Адреса в банках: *а* — последовательное распределение; *б* — чередование

На рис. 7.6, *а* показано последовательное распределение адресов оперативной памяти между четырьмя банками ($m = 2$) в предположении, что каждый банк имеет объем n байт. В этом случае обращение к соседним байтам памяти приходится на

один и тот же банк, а значит, можно организовать только последовательную их выборку.

На рис. 7.6, *б* показано распределение адресов между этими же четырьмя банками, но уже с их чередованием. Байт с номером *j* помещается в банк с номером *j* mod 4. Чередование адресов используется во многих типах микросхем памяти как дополнительный способ повышения производительности.

7.4.2. Микросхемы FPM DRAM

Ускорить операции выборки из ячеек микросхемы или записи в них можно за счет уменьшения длительности фазы адреса. Если зафиксировать некоторую строку микросхемы (или банка), а затем выбирать ячейки только из этой строки, то можно сэкономить время за счет отказа от формирования адреса строки, одного и того же у всех обрабатываемых ячеек. Выделенную для многократной выборки строку микросхемы принято называть *страницей памяти*, а микросхемы, работающие по описанному принципу, относят к типу **FPM DRAM** (от Fast Page Mode DRAM — динамическая память с быстрым страничным режимом). Микросхемы этого типа были разработаны к 1982 г.

Реализуется быстрый страничный режим следующим образом. В стандартном режиме после считывания или записи кода из ячейки микросхемы (или банка) восстанавливается исходный уровень обоих сигналов, RAS# и CAS#. В памяти со страничным доступом после выбора страницы (строки матрицы) сигнал RAS# остается на низком уровне в течение нескольких установок адресов столбцов и соответствующих им сигналов CAS#. Такое состояние микросхемы называется *открытой страницей*. Таким образом, в первом цикле памяти устанавливаются оба адреса и оба сигнала, RAS# и CAS#, а в каждом последующем цикле в фазе адреса устанавливается только сигнал CAS#, что почти вдвое уменьшает время доступа для каждого последующего цикла памяти. Это позволяет в ускоренном режиме выполнять обработку ячеек, которые находятся в строке, выделенной по сигналу RAS#. Если для стандартной микросхемы DRAM цикл памяти имеет вид 5–5–5–5, то для микросхем типа FPM DRAM эта диаграмма имеет вид 5–3–3–3. Обратите внимание на то, что длительность первого цикла памяти (5 тактов) больше, чем длительность всех следующих за ним циклов, работающих с той же самой строкой микросхемы (3 такта). Таким образом, если для выборки четырех последовательных ячеек в случае стандартной микросхемы требуется 20 тактов, то для выборки в страничном режиме нужно всего 14.

Заметим, что выигрыш может получиться только в том случае, если программа действительно запрашивает данные из ячеек микросхемы (банка), находящихся в одной и той же строке микросхемы. Если программа запрашивает данные случайным образом из разных строк, то микросхема FPM DRAM не имеет преимуществ по сравнению со стандартной микросхемой DRAM. Но поскольку во многих программах циклы обработки элементов массивов, размещенных в соседних полях памяти, встречаются довольно часто, микросхемы FPM DRAM в целом эффективнее микросхем DRAM. Некоторые разновидности микросхем FPM

DRAM поддерживают возможность чередования адресов, что дополнительно улучшает их характеристики.

Характерное время доступа для микросхем FPM DRAM примерно 60 нс. А длительность циклов памяти в уже открытой странице сокращается до 30 нс. Микросхемы FPM DRAM могут работать на тактовых частотах системной шины до 66 МГц. В настоящее время они считаются устаревшими.

7.4.3. Микросхемы EDO DRAM

В 1995 г. был предложен вариант улучшения эффективности микросхем FPM DRAM. Это улучшение обусловлено совмещением во времени считывания данных с выхода микросхемы и фазы формирования адреса следующего столбца в одном и том же цикле памяти. Для организации такого совмещения на выходе микросхемы был установлен дополнительный регистр-защелка, выполненный на обладающих большой скоростью триггерных вентилях. После выставления сигнала CAS# данные из текущей ячейки с большой скоростью записываются в этот регистр, а затем, пока они со стандартной скоростью передаются из регистра на шину данных, на адресную шину микросхемы выставляется адрес следующего столбца. Таким образом, в цикле памяти фактически организован конвейер, когда разные устройства одновременно выполняют разные этапы в некоторой операции. В данном случае это контроллер памяти, выставляющий адрес следующего столбца, и шина данных, одновременно с контроллером считывающая код из регистра-защелки.

Микросхемы, работающие в описанном режиме, относят к типу **EDO DRAM** (от Extended Data Out DRAM — расширенное время удержания данных на выходе). Иногда этот режим называют **гиперстраничным** и обозначают **HPM DRAM** (от Hyper Page Mode DRAM). Цикл памяти этого режима имеет структуру 5–2–2–2, и, таким образом, четыре последовательных считывания требуют всего 11 тактов. В режиме EDO DRAM примерно на 15 % по сравнению с FPM DRAM ускоряется процесс считывания последовательных элементов массивов. При случайной адресации такая память ничем не отличается от стандартной.

Характерное время доступа у микросхем EDO DRAM 50–70 нс, а длительность цикла при открытой странице доходит до 20 нс. Они могут работать на тактовых частотах до 66 МГц и также считаются устаревшими для использования в качестве оперативной памяти. Однако эти микросхемы могут использоваться в качестве собственной памяти некоторых внешних устройств (например, лазерных принтеров).

7.4.4. Микросхемы BEDO DRAM

Следующим шагом в повышении эффективности микросхем стало появление так называемого *пакетного*, или *блочного*, режима работы, который является усовершенствованием режима EDO DRAM. Блочный режим заключается в том, что во время чтения одной ячейки вместе с ней читает еще три, расположенных в той

же строке, независимо от наличия или отсутствия запроса на них. Такая группа байтов, передаваемых из банка памяти, считается **пакетом**, или **блоком** — отсюда название режима работы. Тип микросхем, в которых реализован такой метод, называется **BEDO DRAM** (от Burst EDO DRAM — расширенное время удержания данных на выходе с блочным доступом, то есть память на основе режима EDO, но работающая не одиночными, а пакетными циклами чтения/записи).

В режимах FPM DRAM и EDO DRAM на каждую из последовательно обрабатываемых ячеек выдается отдельный адрес столбца и отдельный сигнал CAS#. В режиме BEDO адрес столбца устанавливается только для первой ячейки. Далее осуществляется простой переход к следующей ячейке в строке микросхемы. Для формирования нужного количества байт в блоке (пакете) в фазе адреса инициализируется специальный счетчик. Формирование счетчика увеличивает время доступа, но это увеличение с лихвой покрывается общим уменьшением времени при выборке пакета.

Цикл памяти этого режима имеет вид 5–1–1–1, а значит, четыре последовательных считывания требуют всего 8 тактов. Характерное время доступа до 70 нс, а длительность цикла в пакете доходит до 15 нс. Допускается тактовая частота до 66 МГц.

7.4.5. Микросхемы SDRAM

Стандартные типы микросхем DRAM, а также микросхемы FPM DRAM, EDO DRAM и BEDO DRAM относятся к асинхронным устройствам. Фаза адреса, моменты выдачи управляющих сигналов и обмен данными у них могут начинаться в произвольные моменты времени. Необходимо только точное соблюдение соотношений между действиями и сигналами по их длительности. В эти временные соотношения включены так называемые *охранные интервалы*, необходимые для стабилизации сигналов и кодов, которые не позволяют достичь теоретически возможного быстродействия памяти.

Поскольку тактовая частота процессора значительно превышает частоту оперативной памяти, процессору приходится совершать пустые циклы в ожидании ответа на свой запрос в оперативную память. Такие циклы ожидания, выполняемые во время обращения к асинхронным микросхемам памяти, существенно замедляют работу системы в целом.

Чтобы избавиться от этого эффекта, были разработаны синхронные типы микросхем памяти, ориентирующиеся на импульсы тактового генератора, к которым жестко привязаны моменты выдачи адресов, управляющих сигналов и кодов данных. В этом случае устраняется необходимость в большинстве циклов ожидания процессора и, кроме того, обеспечивается экономия времени за счет отсутствия охранных интервалов.

Появившиеся к 1997 г. микросхемы памяти, работающие в привязке к тактовым импульсам, называются **SDRAM** (от Synchronous DRAM — синхронизированная динамическая память). Ориентация на тактовые импульсы в синхронизированной

памяти позволяет организовать совмещение во времени для значительно большего количества этапов цикла памяти, запустив их конвейерное выполнение, а также увеличить продолжительность совмещений во времени. Помимо синхронного метода чтения/записи в микросхемах SDRAM используется расслоение памяти на независимые банки, что позволяет совмещать выборку из одного банка с установкой адреса в другом банке. Кроме того, SDRAM поддерживает блочный обмен, характерный для микросхем BEDO DRAM, но стандартный пакет банка микросхемы SDRAM длиннее, он состоит из 32 байтов.

Эффективность микросхем SDRAM значительно выше, чем EDO или BEDO. Хотя цикл памяти имеет вид 5–1–1–1, то есть четыре операции проходят так же, как и у BEDO, за 8 тактов, SDRAM имеет длительность цикла 10 нс.

В некоторых разновидностях маркировки (спецификации) микросхем SDRAM указывается частота системной шины, то есть частота, на которой микросхемы могут работать. Например, спецификация PC66 означает, что допустимая частота системной шины у данной микросхемы памяти от 66 до 83 МГц, спецификация PC100 соответствует частоте системной шины 100–120 МГц, а спецификация PC133 — частоте системной шины до 150 МГц. Время доступа у указанных микросхем составляет 7–9 нс. Они обладают скоростью обмена от 256 до 1000 Мбайт/с. А стандартные объемы одной микросхемы памяти типа SDRAM составляют 128, 256 и 512 Мбайт.

7.4.6. Микросхемы DDR DRAM и RDRAM

Относительно новый тип микросхем памяти **DDR SDRAM** (Double Data Rate SDRAM — синхронизированная динамическая память с удвоением данных) появился вследствие улучшений архитектуры SDRAM, поэтому встречается и другое его название — SDRAM II. Удвоение скорости достигается не только за счет увеличения тактовой частоты, но и за счет передачи данных два раза за один цикл памяти: первый раз передача происходит в начале цикла, а второй — в его конце. Следовательно, за один такт передается двойной объем данных. Время доступа составляет 5–6 нс, а допустимая частота системной шины 800–1200 МГц. Отметим, что дальнейшим развитием DDR SDRAM является стандарт DDR2 SDRAM, в котором передача данных осуществляется уже четыре раза за цикл.

Потребителя в большей степени интересует пропускная способность микросхем памяти, которая в связи с новым способом работы микросхем стала зависеть от тактовой частоты более сложным образом. Поэтому у микросхем типа DDR SDRAM в маркировку вынесена не тактовая частота, а скорость обмена: например, микросхема PC2100 имеет скорость обмена равную 2100 Мбайт/с, а PC3200 — 3200 Мбайт/с. В качестве дополнительного параметра маркировки микросхем оперативной памяти используется наличие в микросхеме уже упоминавшейся подсистемы контроля и коррекции ошибок ECC.

Не так давно появилась еще одна разновидность синхронизированных модулей памяти с названием **RDRAM** (Rambus DRAM — от названия фирмы-производителя Rambus, Inc.). В микросхемах этого типа динамической памяти, в частности,

предусмотрена 16-разрядная шина передачи данных, при этом данные, так же как у микросхемы DDR DRAM, передаются по два пакета за такт. Время доступа составляет 4 нс, а частота системной шины – до 800 МГц. Микросхемы памяти типа RDRAM имеют типичный объем 2–4 Гбайт и скорость обмена до 6000 Мбайт/с.

Контрольные вопросы и упражнения

1. Дайте определения технических характеристик микросхем памяти.
2. Опишите принцип действия микросхем статической памяти.
3. Опишите принцип действия микросхем динамической памяти.
4. Что представляет собой регенерация памяти? Как и когда она производится?
5. Дайте сравнительную характеристику микросхем статической и динамической памяти.
6. Опишите устройство микросхемы памяти.
7. Поясните смысл терминов «ячейка микросхемы», «структура микросхемы», «банк памяти», «модуль памяти».
8. Охарактеризуйте существующие конструкции модулей памяти.
9. Как физический адрес байта преобразуется в адрес ячейки микросхемы?
10. Что представляет собой цикл памяти? Как изображаются циклы памяти?
11. Опишите условные приемы изображения временны́х диаграмм циклов памяти.
12. Изобразите временну́ю диаграмму цикла чтения и опишите последовательность действий, выполняемых во время цикла.
13. Охарактеризуйте фазы адреса, данных и восстановления.
14. Поясните смысл характеристик «быстродействие памяти», «время доступа к памяти», «длительность цикла памяти».
15. Опишите метод расслоения памяти. Что дает его использование?
16. Охарактеризуйте особенности микросхем памяти типа FPM DRAM.
17. Охарактеризуйте особенности микросхем памяти типа EDO DRAM.
18. Чем отличается расслоение памяти от приема, используемого в микросхемах EDO DRAM?
19. Охарактеризуйте особенности микросхем памяти типа BEDO DRAM.
20. Охарактеризуйте особенности микросхем памяти типа SDRAM.
21. Охарактеризуйте особенности микросхем памяти типа DDR DRAM.

Глава 8
Многоуровневая организация памяти

В главе 4 учебника были рассмотрены структура и функции двух основных уровней памяти — регистров процессора и оперативной памяти. Упоминался еще и третий уровень, на котором находятся внешние запоминающие устройства, играющие роль информационных складов. Эти уровни выполняют в организации вычислений самостоятельные функциональные роли, и присутствие в составе компьютера по крайней мере первых двух уровней памяти является обязательным. Запоминающие устройства компьютера различных уровней существенно отличаются друг от друга своими базовыми характеристиками, такими как объем, скорость обмена и стоимость.

Этот набор характеристик внутренне противоречив. В самом деле, как правило, увеличение объема памяти сопряжено с увеличением стоимости и уменьшением скорости выборки, так как выбирать нужно из большего количества единиц памяти. Борьба за снижение стоимости приводит к ухудшению остальных характеристик.

Кроме противоречий между базовыми характеристиками одного и того же уровня, имеются противоречия между различными уровнями памяти, важнейшим из которых является существенная разница в их быстродействии, и по мере совершенствования аппаратных средств компьютера эта разница только нарастает.

Чтобы разрешить эти противоречия и повысить общую эффективность системы, память компьютера реализована в виде многоуровневой структуры, которая включает некоторое количество дополнительных уровней, сглаживающих разницу между соседними основными уровнями.

Кратко охарактеризуем свойства используемых в настоящее время уровней памяти компьютера.

- **Регистровая память процессора.** Используется для промежуточного хранения данных в процессе их обработки. Энергозависимый вид памяти. Обладает наиболее высоким быстродействием, поскольку работает на тактовых частотах

процессора. Время доступа составляет десятые доли наносекунд. Объем регистров — десятки и сотни машинных слов.

- **Кэш-память.** Дополнительный уровень памяти, играющий роль буфера между оперативной памятью и процессором. Служит для сглаживания разницы в скоростях их работы. Энергозависимый вид памяти. За исключением регистрового, это самый быстрый тип памяти, реализуемый на *статических* микросхемах памяти. Время доступа составляет единицы наносекунд. Объем кэша может достигать нескольких мегабайт. Отличается относительно высокой стоимостью и высоким энергопотреблением.
- **Оперативная память.** Используется для хранения выполняющихся программ и необходимых им данных. Энергозависимый вид памяти. Время доступа составляет десятки наносекунд, а объем — несколько гигабайт. Более дешевая, чем кэш-память, так как реализуется с помощью динамических микросхем. Высокая надежность хранения данных и программ, расчетная вероятность ошибки — одна в десять лет.
- **Внешняя память** — магнитные и оптические диски. Служит информационным складом. Самый медленный уровень памяти, время доступа порядка 30 000 000 нс. Самая дешевая и самая емкая — объем современных дисков доходит до десятков терабайт.

Анализируя приведенный список, можно заметить, что уровни памяти компьютера как бы образуют пирамиду, в основании которой находится внешняя память, а на вершине — регистровая память. Уровень, лежащий в основании, — самый медленный, но зато обладает огромным объемом. По мере продвижения к вершине пирамиды характерный объем уменьшается, а скорость обмена увеличивается, достигая максимального значения у регистрового уровня.

Кроме перечисленных выше уровней памяти в компьютере используются еще несколько разновидностей запоминающих устройств, которые играют, в общем-то, вспомогательную, но довольно важную роль. Это буферная, постоянная и полупостоянная память.

Буферная память используется в адаптерах, контроллерах, портах и т. д. для временного хранения данных в процессе ввода/вывода с целью организации асинхронной работы внешних устройств или сглаживания разницы в скоростях работы устройств, между которыми осуществляется обмен. Это энергозависимый вид памяти. Различные устройства имеют разную по скорости и объему буферную память. Самым высоким быстродействием, сравнимым с быстродействием кэш-памяти, и объемом до десятков мегабайт обладает буферная память контроллеров дисплеев.

Постоянная память (**ПЗУ** — постоянное запоминающее устройство, или **ROM**, от Read Only Memory — память только для чтения) используется для энергонезависимого хранения важной системной информации. В постоянной памяти всегда находится часть операционной системы, которая называется **BIOS** (от Base Input/Output System — базовая система ввода/вывода). BIOS — это набор программ проверки и обслуживания аппаратуры компьютера, который обеспечивает

также выполнение простейших операции ввода с клавиатуры, вывода на дисплей и т. д. Постоянная память допускает только считывание. Типовое значение объема ПЗУ до 256 Кбайт при невысокой скорости обмена и времени доступа более 100 нс. Для повышения производительности содержимое постоянной памяти копируется в оперативную, и в работе используется только копия BIOS, которую часто называют **теневой** памятью. В последние годы постоянная память вытесняется другими видами энергонезависимой памяти, такими как **флэш-память** (от flash — вспышка, мгновение).

Основными разновидностями постоянной памяти являются:

- **PROM** (от Programmable ROM — однократно программируемые ПЗУ). Эта разновидность памяти отличается тем, что для записи программ в микросхемы памяти используется специальное устройство. Модули памяти вынимаются из корпуса компьютера и вставляются в это устройство, в котором в модули заносится новое содержимое. Затем модули памяти вставляются назад в компьютер. Этот вид памяти считается устаревшим;
- **EPROM** (от Erasable Programmable ROM — перепрограммируемые ПЗУ). Микросхемы отличаются тем, что можно *неоднократно* стирать их содержимое и заносить новое. Удаление кодов из микросхем памяти осуществляется с помощью ультрафиолетового облучения в течение нескольких минут через специальные окошки, которые после облучения вновь заклеиваются. Перепрограммирование микросхем можно выполнять прямо на компьютере с помощью специально подключаемого устройства. Также устаревший вид памяти;
- **EEPROM** (от Electrically Erasable Programmable ROM — электрически стираемые перепрограммируемые ПЗУ). Стирание осуществляется прямо в компьютере с помощью высокого (порядка 30 В) напряжения;
- **FEPROM** (от Flash EPROM), или просто флэш-память. Объединяет достоинства ROM и RAM, то есть допускает не только чтение, но и запись в обычном режиме работы компьютера, так же как и микросхемы DRAM, но при выключении электропитания содержимое памяти не уничтожается. В отличие от DRAM потребляет энергию не в течение всего времени работы, а только в периоды чтения/записи.

Полупостоянная память обеспечивает хранение относительно небольшого объема важных параметров конфигурации компьютера (характеристики процессора, системного диска и т. д.) и операционной системы даже при отключенном электропитании. Для поддержания внутреннего состояния такой памяти при отключенном электропитании используется батарейка или аккумулятор. Реализуется полупостоянная память с помощью **CMOS-вентилей** (от Complimentary Metal Oxide Semiconductor — комплементарный металл-оксид-полупроводник, соответствующее русское сокращение — **КМОП**). К полупостоянной памяти относится также **CMOS RTC** (от CMOS Real Time Clock — память часов реального времени) и **ESCD** (от Extended Static Configuration Data — расширенные статические данные конфигурации). Память CMOS RTC используется для хранения

системных часов и календаря, а память ESCD – для хранения параметров устройств с автоматическим конфигурированием, которые обычно называют **PnP-устройствами** (от Plug and Play – вставь и работай).

Отметим, что под **конфигурированием** устройства понимается осуществляемый во время его подключения к компьютеру подбор параметров и режимов работы, номера прерывания и т. д., которые обеспечивают нормальное взаимодействие с остальными устройствами компьютера и с операционной системой. Такое конфигурирование, как правило, осуществляется один раз, а затем выбранные параметры должны сохраняться даже при отключенном электропитании. Объем полупостоянной памяти составляет несколько сотен байт, а время доступа к ней превышает 100 нс.

Дополнительный материал по обсуждавшимся в главе вопросам можно найти в изданиях [18], [30], [34].

Контрольные вопросы и упражнения

1. Для чего потребовалась реализация многоуровневой подсистемы памяти компьютера?
2. Охарактеризуйте основные уровни памяти компьютера.
3. Охарактеризуйте вспомогательные уровни памяти компьютера.

Глава 9
Кэш

В любом случае взаимодействия с оперативной памятью процессор не может получить или передать данные со скоростью, превышающей возможности памяти. На современном уровне развития вычислительных систем скорость работы процессора в десятки и даже сотни раз выше скорости работы микросхем памяти. Поэтому во время обмена с оперативной памятью процессор теряет много тактов на ожидание данных из памяти.

В 1960-е гг. в составе компьютеров появилась так называемая **сверхоперативная память**, существенно повысившая общую производительность вычислительных систем. В персональных компьютерах аналогичную функциональную роль стал играть появившийся в 1985 г. высокоскоростной буферный уровень памяти, который назвали **кэш-памятью** (от cache — запас, тайник, наличные в кармане). Точнее всего способ использования этого уровня памяти характеризует сленговый перевод слова cache — карманные деньги, то есть находящиеся под рукой и которые быстро, в любой момент можно достать и использовать для любых нужд.

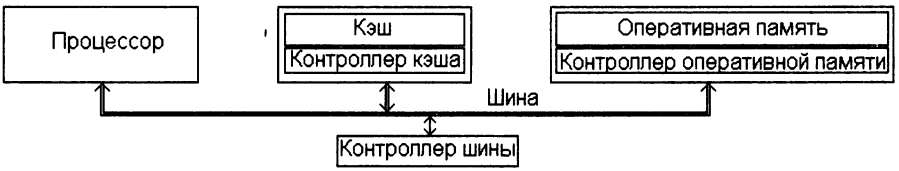


Рис. 9.1. Упрощенная схема взаимодействия процессора, кэша и оперативной памяти

Объем кэша значительно меньше, чем объем оперативной памяти, а скорость больше, чем у микросхем динамической памяти, и примерно такая же, как у процессора. В общем случае кэш помещается между оперативной памятью и процессором (рис. 9.1).

Экспериментально установлено, что после обработки некоторых данных процессор с большой вероятностью обращается к тем же самым данным или к находящимся

в непосредственной близости от них. Это наблюдение, которое называется **принципом пространственной локализации**, уже обсуждалось при изучении пакетного режима функционирования микросхем памяти. Аналогом принципа пространственной локализации является также экспериментально полученный **принцип временно́й** локализации, который утверждает, что программе в ближайшее время, вероятнее всего, потребуются данные, которые недавно или только что были использованы.

С учетом принципов локализации данные, затребованные процессором, а также некоторая группа находящихся рядом кодов не только передаются в процессор, но и автоматически заносятся в кэш. Когда процессору нужны какие-либо данные, он вначале «заглядывает» в кэш «в надежде» обнаружить их там. Если нужные данные действительно находятся в кэше, то они с высокой скоростью передаются в процессор. Эта ситуация носит название **попадание в кэш**. Если данные в кэше отсутствуют, то говорят, что имеет место **промах кэша**, и процессор вынужден обращаться в оперативную память.

Если при попытке записи новых данных в кэш он оказывается до конца заполненным, необходимое для записи место освобождается за счет удаления из кэша некоторой группы уже находящихся в нем данных. Удаляться могут данные, к которым было меньше всего обращений, или, в соответствии с принципом временно́й локализации, данные, которые дольше всего не использовались процессором. Применение описанного подхода приводит к тому, что по мере выполнения программы в кэше скапливаются наиболее часто используемые данные. Такое накопление принято называть **кэшированием**.

Очевидно, что при попадании в кэш происходит существенное ускорение работы системы. Количественно оценить общее ускорение от использования кэша можно следующим образом. Пусть время доступа к микросхемам оперативной памяти равно $t_о$, время доступа к микросхемам кэша — $t_к$, h — доля попаданий в кэш от общего количества обращений к памяти. Тогда доля промахов в общем количестве обращений составляет $1 - h$, а среднее время доступа при наличии кэша $t = t_о(1 - h) + t_к h$. Например, для $t_о = 10$ нс, $t_к = 1$ нс и $h = 0{,}75$ получим $t = 3{,}25$ нс — более чем в три раза лучше, чем при отсутствии кэша.

Может показаться, что промахи кэша замедляют работу, так как процессор, безрезультатно «заглянув в кэш», делает лишнюю операцию. Однако эту проблему можно обойти, организовав параллельное обращение в оперативную память и в кэш. Если данные в кэше обнаружатся, то выборка из памяти прекращается досрочно. Если же данные в кэше отсутствуют, то выборка из памяти завершается стандартным образом. В любом случае дополнительные временные затраты очень малы.

9.1. Механизмы работы кэша

К настоящему времени разработано несколько различных вариантов организации кэша. Рассмотрим наиболее известные из них. В общем случае кэш состоит из области хранения данных, в которой находятся копии некоторых полей

оперативной памяти — собственно кэш-памяти, или **памяти данных** (рис. 9.2), и области, которая содержит управляющую информацию в виде набора признаков, описывающих находящиеся в кэше копии. Эту область кэша называют **памятью тегов** (от tag — ярлык, этикетка, признак).

9.1.1. Кэш прямого отображения

Исторически первым и наиболее простым является кэш **прямого отображения**. Память для хранения копий делится на элементы, которые принято называть **строками кэша**. Строки кэша могут быть различной длины. Довольно часто используется длина 32 байта, соответствующая стандартно передаваемой из оперативной памяти в процессор группе байтов (пакету). Роль строки кэша состоит в том, что данные из оперативной памяти дублируются в кэш сразу целой строкой, которая содержит запрошенное процессором поле памяти. Таким образом, в кэш попадает не только копия требуемого поля, но и копия группы байтов, расположенных рядом с ним.

Количество строк в кэше зависит от имеющегося в компьютере физического объема кэша и от длины его строк. Так, например, кэш объемом 4 Кбайт может состоять из 128 тридцатидвухбайтных или из 256 шестнадцатибайтных строк. Встречаются и другие варианты. Каждой строке кэш-памяти однозначно соответствует элемент памяти тегов, который содержит значения рассматриваемых далее признаков этой строки.

Для получения простого и быстродействующего механизма поиска контроллер кэша рассматривает всю имеющуюся оперативную память как совокупность страниц, которые равны по объему кэшу и также состоят из строк. Простота дублирования и поиска в кэше обеспечивается тем, что копия любой строки из любой страницы всегда занимает в кэше точно такое же положение, что и оригинал на соответствующей странице оперативной памяти. Это значит, что смещение копии строки относительно начала кэш-памяти всегда равно смещению оригинала строки относительно начала ее страницы в оперативной памяти (рис. 9.2). Таким образом, страница оперативной памяти как бы полностью отображается на память данных в кэше — отсюда и название «кэш прямого отображения».

Каждая страница оперативной памяти имеет номер, который принято называть **тегом**. Когда копия строки из какой-либо страницы попадает в кэш, ее тег занимает одноименное поле соответствующего элемента памяти тегов.

Для определения положения запрошенного процессором байта (или поля памяти) контроллер кэша выделяет в его физическом адресе три участка. Находящиеся в этих участках коды считаются номером страницы оперативной памяти, номером строки на странице и номером байта в строке. Разрядность участков зависит от длины физического адреса, количества страниц и длины строк.

В качестве примера для обсуждения будем считать, что кэш объемом 4 Кбайт состоит из 128 тридцатидвухбитовых строк, а физический адрес состоит из 20 битов. Тогда для определения номера байта в строке нужно 5 битов ($2^5 = 32$), а для задания строки на странице — 7 битов ($2^7 = 128$). Из 20 битов адреса на номер страницы

остается 8 битов, с помощью которых можно перенумеровать $2^8 = 256$ страниц. Таким образом, рассматриваемый кэш может использоваться для кэширования оперативной памяти объемом 4 Кбайт · 256 = 1 Мбайт.

Рис. 9.2. Кэш прямого отображения

Распределим разряды физического адреса следующим образом. В пяти младших битах (4–0) будем кодировать номер байта в строке, следующие 7 битов (11–5) займем номером строки, а последние 8 битов (19–12) пусть содержат номер страницы (рис. 9.2, *внизу*). Допустим, что процессору потребовался байт оперативной памяти с физическим адресом $18A72_{16}$ = 0001 1000 1010 0111 0010$_2$. После выделения контроллером кэша в этом адресе соответствующих участков 00011000 | 1010011| 10010$_2$ окажется, что нужный байт расположен на странице оперативной памяти с номером 18_{16} в строке с номером 53_{16}, а его номер в строке 12_{16}. Во время передачи байта из оперативной памяти в процессор вся содержащая его 32-битная строка попадает и в кэш, контроллер которого запишет ее копию в строку с номером 53_{16}. Номер страницы оперативной памяти (18_{16}), из которой была выбрана строка, заносится в поле тега соответствующего строке элемента памяти тегов.

Если теперь процессору потребуется какой-либо байт из оперативной памяти, то кроме обращения в память процессор направляет запрос и контроллеру кэша. Далее по описанной схеме контроллер находит номер строки кэша, в которой может находиться нужный байт. Если тег строки кэша совпадает с номером страницы физического адреса, то обращение в оперативную память прекращается, а нужный байт передается в процессор из кэша — говорят, что имеет место попадание в кэш. Такая ситуация наблюдается, если, например, процессор запросит данные из физического адреса $18A74_{16}$. А вот для адреса $20A72_{16}$, который обращается к байту в строке с тем же номером 53_{16}, но на другой странице, тег 18_{16} не совпадает с ее номером 20_{16}. В этом случае имеет место промах кэша. Тогда процессор осуществит выборку из оперативной памяти. Кроме того, произойдет

замена всей строки кэша с номером 53_{16}, а также соответствующего строке элемента памяти тегов. В таких случаях говорят, что произошло **вытеснение (выталкивание)** строки из кэша.

Основным достоинством кэш-памяти прямого отображения является высокая скорость определения попадания или промаха. Кроме того, весьма просто решаются основные задачи управления кэшем, в частности, задача вытеснения строки, когда кэш уже заполнен.

9.1.2. Многовходовый ассоциативный кэш

Кэш прямого отображения имеет и существенный недостаток. В него могут попасть только копии строк с разными номерами (или, что то же самое, с разными смещениями относительно начала страницы) из одной и той же или из разных страниц оперативной памяти. Копии строк, которые имеют один и тот же номер, но расположены на разных страницах, не могут одновременно находиться в кэше, так как при любой попытке обратиться к строке на другой странице копия строки из первой страницы заменяется в кэше. Например, в кэше не могут одновременно находиться копии строки с номером 53_{16} из 18_{16}-й и 20_{16}-й страниц памяти, так как обращение к 20_{16}-й странице вытеснит из кэша копию этой строки из 18_{16}-й страницы. Такая замена требует дополнительных временны́х расходов, и при частых заменах весь выигрыш от наличия кэша может исчезнуть, а то и заменится снижением общей эффективности из-за необходимости вместо байта или слова считывать из оперативной памяти каждый раз целую строку.

Размещение в кэше двух и более копий «однономерных» строк из разных страниц оперативной памяти допускается в так называемом **ассоциативном кэше с множественным доступом**, или **многовходовом ассоциативном** кэше. Иногда такой вариант организации называют **наборно-ассоциативным кэшем**. Количество копий «однономерных» строк из разных страниц, которые могут быть помещены в кэш, может отражаться в названии кэша: ***n*-входовый** (или ***n*-связный**) **ассоциативный** кэш, например двухвходовый кэш. Понятно, что n-входовый кэш допускает наличие в нем n копий «однономерных» строк. В настоящее время наиболее распространены четырех- и восьмивходовые ассоциативные кэши.

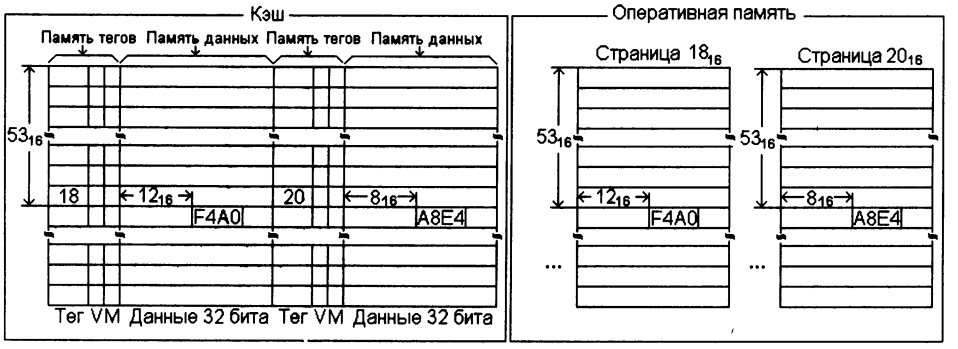


Рис. 9.3. Двухвходовый ассоциативный кэш

На рис. 9.3 изображен двухвходовый ассоциативный кэш. По сути дела он представляет собой два экземпляра кэша прямого отображения, которые принято называть **банками кэша**. Каждый из банков ассоциативного кэша может содержать копию строки с одним и тем же номером, но из разных страниц памяти. Кроме того, допускается независимый и одновременный просмотр тегов из разных банков. В изображенной на рисунке ситуации в кэше находятся копии двух строк с номером 53_{16}: одна из 18_{16}-й, а вторая — из 20_{16}-й страницы.

9.1.3. Ассоциативная память

Отметим, что ассоциативный кэш является разновидностью **ассоциативной памяти**. Выборка кода из обычной памяти, которую называют также **адресуемой**, осуществляется с помощью задания адреса, никак не связанного (не ассоциирующегося) со значением кода. В ассоциативной памяти выборка кода связана, т. е. ассоциируется (отсюда и название разновидности памяти) с его значением или со значением некоторой его части, которую принято называть **ключом поиска**, или просто **ключом**.

В общем случае в ассоциативной памяти каждый записываемый код сопровождается дополнительным уникальным (то есть не повторяющимся нигде в другом месте) кодом ключа. Можно даже считать, что ключ, в отличие от адреса, является неотделимой составной частью кода данного. Например, в случае ассоциативного кэша таким ключом поиска является тег страницы, сопровождающий копию любой строки в кэше. Для выполнения чтения значение ключа кода в памяти сравнивается с заданным значением ключа поиска. В случае совпадения сравниваемых значений код считывается из памяти.

Реализуется ассоциативная память значительно сложнее, чем адресная память, поскольку необходимо организовать аппаратный просмотр некоторого количества полей, занятых ключами. Такой просмотр может быть организован как последовательный, когда коды по очереди сравниваются друг с другом, или как параллельный, когда выполняется поразрядное сравнение сразу для всех ключей. Понятно, что второй способ выполнения поиска гораздо быстрее первого, но микросхемы памяти получаются более дорогими. Ассоциативная память используется в основном для решения задач, требующих быстрого поиска в относительно небольших объемах данных.

9.1.4. Управление ассоциативным кэшем

Для выявления наличия нужных данных в ассоциативном кэше уже недостаточно анализа физического адреса и сравнения тега с номером строки. В нем следует просмотреть n элементов памяти тегов в n различных банках кэша, причем лучше организовать их параллельный просмотр. Если тег в одном из банков совпадает с номером страницы из физического адреса, то поиск завершается попаданием в кэш. Если нет совпадения ни в одном из банков, фиксируется промах кэша.

Сложности возникают также при определении замещаемой строки в случае, когда нужная строка занята во всех банках. Пусть, например, в изображенной на

рис. 9.3 ситуации процессором вызывается строка 53_{16} из страницы с номером 21_{16}. Оба банка заняты, поэтому нужно решить, какую из копий строк (из 18_{16}-й или из 20_{16}-й страницы) следует заменить копией строки из 21_{16}-й страницы. При использовании кэша прямого отображения такой проблемы нет — занятая позиция кэша перезаписывается при любом новом поступлении на эту позицию. В многовходовом ассоциативном кэше кандидатов на вытеснение несколько, и желательно выбрать вытесняемую копию так, чтобы минимизировать количество промахов кэша. Для такого выбора применяется несколько правил, которые обычно называют **алгоритмами замещения**. Чаще всего используется алгоритм **LRU** (от least recently used — наиболее давно использованный), в соответствии с ним из кэша вытесняется строка, к которой дольше всего не было обращений. Для его реализации может быть, например, составлен связный список копий «однономерных» строк из разных банков. При каждом обращении в кэш этот список модифицируется так, что в его начало попадает наиболее часто (а в конец — наименее часто) используемый элемент. При возникновении необходимости из кэша вытесняется последний элемент списка. Существуют и другие варианты реализации этого алгоритма.

9.2. Многоуровневый кэш

В современных вычислительных системах кэш, как правило, имеет сложную многоуровневую структуру. Это значит, что в состав системы включается несколько различающихся объемом и быстродействием устройств, выполняющих функции кэша.

Самый быстродействующий, но при этом самый маленький по объему уровень кэша встраивается в микросхему процессора. Он обозначается **L1** и называется **внутренним**, или **первичным**, кэшем. В настоящее время внутренний кэш обычно реализуется в виде двух независимых банков объемом 8–32 Кбайт каждый. Один банк внутреннего кэша используется только для хранения кодов данных, а второй — только для хранения кодов команд. Такую структуру называют **разделенной**, или **гарвардской**, архитектурой кэша, поскольку впервые она появилась в машине «Марк 3», созданной Говардом Айкеном в Гарвардском университете. Раздельная структура кэша имеет ряд преимуществ. Во-первых, банки кэша допускают одновременное и независимое выполнение операций считывания/записи через отдельные каналы доступа к оперативной памяти. Это примерно вдвое увеличивает пропускную способность кэша. Во-вторых, разделенный кэш допускает упрощенную организацию банка с кодами команд, поскольку во время выполнения программы они обычно не меняются, в то время как находящиеся в кэше коды данных могут изменяться.

Отметим, что в связи с включением в микросхему процессора устройств, не связанных с его основными функциями, введено понятие **ядро процессора**, к которому относится только часть его микросхемы, содержащая арифметико-логическое устройство, устройство управления, регистры и т. д., то есть те элементы, которые и составляют собственно процессор. Любые дополнительные элементы,

включаемые в микросхему процессора для повышения эффективности системы, такие как, например, кэш первого уровня, не входят в ядро процессора.

Следующий уровень кэша обозначается **L2** и называется **внешним**, или **вторичным**. Он реализуется в виде отдельной микросхемы статической памяти объемом 256–512 Кбайт. Если в системе используется всего два уровня кэша, то внешний кэш может иметь объем 2–4 Мбайт. Обычно второй уровень кэша содержит одновременно как коды команд, так и коды данных. Такую структуру называют **объединенной**, или **принстонской**, архитектурой кэша.

Довольно часто компьютеры имеют еще один внешний кэш — кэш третьего уровня **L3**. Его объем доходит до 16–32 Мбайт, а быстродействие ниже, чем у кэша первого и второго уровней, но все-таки существенно выше, чем у микросхем динамической оперативной памяти. Обычно содержимое кэша первого уровня целиком находится в кэше второго уровня. А при наличии кэша третьего уровня содержимое двух верхних уровней целиком помещается в третий.

Управлять многоуровневым кэшем значительно сложнее, чем одноуровневым, но получаемое от увеличения количества уровней повышение общей эффективности вычислительной системы привело к повсеместному использованию такой архитектуры.

9.3. Когерентность кэша

Выполнение операций считывания кодов при наличии кэша не вызывает особых проблем. Сложности начинаются при выполнении операций записи. Они вызваны наличием нескольких копий (хотя бы двух) одного и того же кода в разных местах. С течением времени одна из копий кода может быть обновлена, заменена другим значением. Тогда возникает несовпадение находящихся в разных местах значений одной и той же величины.

Вообще говоря, это общая для всей информатики (и не только информатики) проблема хранения информации. Данные приходится дублировать в разных случаях и с разными целями. Например, в базах данных для обеспечения надежности хранения создается несколько архивных копий базы, которые к тому же принято размещать в разных местах. При необходимости внести в базу какие-либо изменения должны быть заменены все ее копии, иначе нарушается целостность данных. Задача поддержания одинаковых значений у кодов, находящихся в оперативной памяти и в кэше, называется **проблемой когерентности**[1] **(достоверности, целостности) кэша**.

Когда исходный код находится в оперативной памяти, а его копия помещена в кэш, возможны два варианта нарушения когерентности. Во-первых, ко многовходовой оперативной памяти может обращаться не только процессор. Например, независимо

[1] Аналог физического понятия когерентности (от лат. cohaerentia), которое означает согласованное протекание во времени нескольких колебательных или волновых процессов. В данном случае имеется в виду согласованное изменение значений кода и всех его копий.

могут выполнять операции записи в память устройства Bus Mastering (жесткие диски по каналам контроллера DMA). В процессе такой записи изменяется только значение кода в оперативной памяти, а его копия в кэше остается в неизменном виде. Во-вторых, процессор может изменить значение в кэше, а код в оперативной памяти при этом оставить без изменения. Для регистрации описанных вариантов несоответствия в элементах памяти тегов предусмотрены флаг достоверности V (от validity — действительность) и флаг модификации M (от modify — изменение). Флаг V принимает значение 1 при записи копии строки в кэш и сбрасывается в 0, если изменился соответствующий код в оперативной памяти. Флаг M, наоборот, принимает значение 0 при записи копии строки в кэш, а устанавливается в 1 при изменении значения в кэше.

В связи с введенным уточнением структуры элемента памяти тегов отметим, что для выборки кода из кэша (чтения кода) оказывается недостаточно только совпадения тега и выделенного из физического адреса номера страницы. Дополнительно требуется, чтобы флаг достоверности V был равен 1, сигнализируя о том, что находящийся в кэше код действителен, то есть совпадает с оригиналом в оперативной памяти. Если это не так, процессору следует выбирать код из оперативной памяти.

Существует несколько способов поддержания когерентности кэша. В схеме **прямой** записи измененный процессором код заменяется в кэше и в оперативной памяти одновременно. Это надежный вариант, в котором несоответствие кодов не возникает. Вместе с тем его использование снижает быстродействие системы.

Более высокой эффективностью обладают схемы, которые выполняют запись в оперативную память с некоторой задержкой во времени относительно момента записи в кэш. При использовании **обратной** записи, которую называют также **буферизованной сквозной записью**, значение передается в память в первом же свободном такте работы процессора. А в схеме **отложенной** записи передача измененного кода в оперативную память выполняется только при окончательном заполнении кэша, то есть когда для помещения в кэш нового значения не оказывается свободной области. При этом в соответствии с алгоритмом LRU в оперативную память переносится наименее часто используемая строка кэша.

Схемы обратной и отложенной записи более эффективны, так как не требуют при выполнении каждой операции записи обращения к оперативной памяти. Но они и более сложны, так как нужно уметь определять, следует ли выталкиваемую из кэша строку переписывать в оперативную память. Для экономии времени строки, флаг M которых равен 0, в оперативную память не дублируются. Кроме того, нужно заботиться о поддержании соответствия содержимого кэша и основной памяти. При использовании этих схем существуют довольно длительные интервалы времени, в течение которых коды в памяти и кэше не когерентны. Следует исключить каким-либо способом доступ к некогерентным данным в течение этих интервалов.

Еще более сложной является проблема когерентности в многоуровневом кэше, в котором необходимо заботиться сразу о нескольких разноуровневых копиях

одного и того же кода, а также в многопроцессорных системах, когда каждый из входящих в систему центральных процессоров имеет собственный кэш и потому может создать собственную копию одной и той же строки из оперативной памяти. Любой из процессоров может изменить значение кода в принадлежащем ему кэше, но копии кода в чужих кэшах ему недоступны. Так возникает еще одна причина появления некогерентных данных.

Для решения этих проблем процессоры Intel соблюдают совокупность правил отслеживания состояния копий, которую называют **протоколом MESI** (от Modified Exclusive Shared Invalid — модифицированная, монопольная, разделяемая, недействительная копия). Обсуждение этого протокола выходит за рамки учебника. Его описание можно найти, например, в [30].

9.4. Микросхемы кэша

Более дешевые микросхемы динамической памяти (DRAM) не могут обеспечить необходимых кэшу скоростных характеристик. Поэтому внешний кэш всегда реализуется на более дорогих микросхемах статической памяти (SRAM), основными разновидностями которой являются **асинхронная (Asynchronous SRAM)**, **синхронная пакетная (Synchronous Burst SRAM)** и **синхронная конвейерно-пакетная (Pipeline Burst SRAM)** статическая память.

Микросхемы асинхронной статической памяти устроены примерно так же, как микросхемы обычной асинхронной динамической памяти, только бит реализуется не периодически подзаряжаемым конденсатором, а описанным в 3.3 полупроводниковым вентилем памяти.

В синхронных видах микросхем пакетной и конвейерно-пакетной памяти передача данных организована так же, как у динамических микросхем BEDO DRAM и SDRAM, — пакетами, что наряду с синхронизацией обеспечивает повышение их эффективности по сравнению с асинхронными микросхемами. В конвейерно-пакетных микросхемах, как и в микросхемах EDO DRAM, введены дополнительные внутренние буферные регистры данных, обеспечивающие конвейерное совмещение во времени действий считывания данных шиной и формирования адреса следующей ячейки. Кроме того, у них, как в микросхемах DDR DRAM, данные передаются на удвоенной скорости. Это обеспечивает время доступа у конвейерно-пакетных микросхем 2–3 нс на частотах более 400 МГц.

Дополнительный материал по обсуждавшимся в главе вопросам можно найти в изданиях [18], [30], [34], [35].

Контрольные вопросы и упражнения

1. Опишите принципы использования кэша в архитектуре компьютера.
2. Сформулируйте принципы заполнения кэша данными.
3. Поясните смысл терминов «попадание в кэш», «промах кэша», «кэширование».

4. Оцените возможный выигрыш от включения кэша в структуру компьютера.
5. Опишите общую структуру кэша.
6. Опишите принцип работы кэша прямого отображения.
7. Опишите принцип действия многовходового ассоциативного кэша.
8. Чем различаются способы выталкивания для многовходового кэша и кэша прямого отображения?
9. Какой способ действий контроллера кэша можно предложить, если запрошенное процессором поле попадает на границу строк, то есть одна часть поля находится в одной строке, а другая часть — в другой?
10. Дайте сравнительную характеристику адресуемой и ассоциативной памяти.
11. Опишите структуру многоуровневого кэша и дайте характеристику его уровням.
12. В чем заключается проблема когерентности кэша? Сформулируйте причины появления некогерентности кэша.
13. Опишите используемые способы поддержания когерентности кэша.
14. Какие микросхемы используются для реализации кэша?

Глава 10
Шины

В 4.1.3 рассмотрен простейший вариант архитектуры компьютера с общей шиной. Впервые 12-битовая общая шина Omnibus появилась в конструкции выпущенной в 1965 г. машины PDP-8. С тех пор шины стали широко использоваться в самых различных моделях компьютеров.

Организация связи между устройствами компьютера по единственной общей шине оказалась довольно дешевым и простым вариантом архитектуры компьютера. Но одна шина могла удовлетворительно справляться с обменом до тех пор, пока быстродействие процессора, микросхем памяти, магнитных дисков и других устройств было относительно низким. По мере роста тактовых частот и объемов передаваемых данных общая шина компьютера стала «узким местом» архитектуры. В связи с этим начались работы по улучшению эффективности шин компьютера.

Основными техническими характеристиками шин являются разрядность, тактовая частота и производительность (скорость обмена, или пропускная способность[1]). Напомним, что разрядность определяется количеством проводов шины, по каждому из которых передается один бит адреса, один бит данных или один управляющий бит. Тактовая частота определяется как количество управляющих работой шины синхроимпульсов в секунду. Производительность равна количеству битов (или байтов), передаваемых по шине в единицу времени. Если, например, 16-битовая шина работает на частоте 100 МГц и выполняет передачу кода за один такт, то ее производительность равна 2 байта · 10^8 тактов/с ≈ 190 Мбайт/с. Не следует путать пропускные способности шины и памяти. Это согласованные, но не обязательно совпадающие величины.

Один из возможных путей повышения производительности шины состоит в увеличении ее разрядности и тактовой частоты. Повышение разрядности приводит

[1] Если быть точным, то пропускная способность — это максимально возможная скорость обмена. Реальная скорость обмена всегда ниже пропускной способности.

к увеличению геометрических размеров шины и, как следствие, к увеличению общей стоимости компьютера. Для уменьшения стоимости и размеров шины были разработаны различные варианты **мультиплексных** шин, в которых нет отдельных линий данных и линий адреса. Коды адресов и данных в таких шинах передаются по одним и тем же линиям. Сначала по шине передается адрес передаваемого кода, затем по тем же самым проводам передаются его биты. Такие шины, однако, обладают относительно невысокой производительностью.

Увеличение тактовой частоты шины может нарушить ее совместимость со старыми устройствами и с существующим программным обеспечением. Кроме того, это увеличение может привести к так называемому **перекосу** шины, который возникает при передаче данных по разным линиям шины с различной скоростью. В силу ряда физических факторов перекос шины вносит ошибки в передаваемые по ее линиям коды, а также является причиной аппаратных сбоев, возникающих при передаче по ней управляющих сигналов. Повышение частоты вызывает особенно сильные помехи в многоразрядных шинах, поэтому одновременно увеличивать и разрядность шин и их частоту невозможно.

Таким образом, разработчикам шин для увеличения их эффективности приходится решать множество задач, которые выдвигают взаимно противоречивые требования. Компромиссные варианты построения шин были получены на пути организации параллельности и конвейеризации в работе шины. Заметим, что в последнее время наметилась тенденция к замене шин с большой разрядностью на шины с меньшей разрядностью, но с более высокой частотой.

10.1. Циклы шин

Как было выяснено ранее, работой шины управляют ее *контроллер* и *арбитр*. Существуют различные конструктивные варианты реализации управляющих функций этих устройств. В частности, функции арбитража могут быть возложены на отдельную микросхему, они могут быть также переданы процессору или же закреплены за контроллером шины.

Шинам компьютера приходится выполнять несколько различных функций. Поэтому каждый вид работы определенным образом стандартизируется в виде отдельного **цикла шины**.

ВНИМАНИЕ

Направленная на выполнение определенной функции шины периодически повторяемая связная последовательность управляющих работой шины сигналов, а также передач кодов по ее линиям называется циклом шины.

В частности, существуют циклы чтения, циклы записи, циклы прерывания и некоторые другие типы циклов шин. Некоторые разновидности циклов рассматриваются в дальнейшем.

По отношению к тактовым синхроимпульсам шины делятся на *синхронные* и *асинхронные*. У синхронных шин все действия цикла шины привязаны к определенным фазам синхроимпульса — началу, середине, концу и т. д. Они имеют строго определенную длительность, а цикл такой шины всегда занимает целое количество тактов. У асинхронных шин такая привязка отсутствует.

10.1.1. Цикл чтения синхронных шин

Рассмотрим изображенную на рис. 10.1 упрощенную временную диаграмму цикла чтения шины. В некоторый момент времени t_a процессор выставляет на адресные линии шины адрес байта, который нужно прочитать в оперативной памяти. Так же, как в описанной в 7.3 диаграмме цикла памяти, на стабилизацию адреса уходит некоторое время. Когда адрес стабилизируется, процессор формирует нулевое значение управляющего сигнала MEMR# (от memory read — чтение из памяти). Этот сигнал сообщает всем присоединенным к шине устройствам, что процессор запрашивает данные из оперативной памяти. Напоминаем: знак # в названии управляющего сигнала означает, что активным, то есть приводящим в действие, является его нулевое значение.

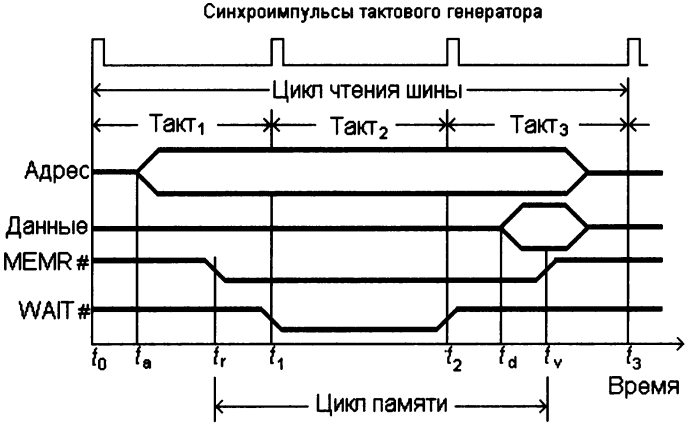


Рис. 10.1. Цикл чтения синхронной шины с одним тактом ожидания

В момент t_r стабилизации указанного сигнала контроллер оперативной памяти получает по шине сигнал о том, что есть запрос на чтение из памяти, а также адрес требуемого байта. С этого момента начинается цикл памяти — контроллер вместе с микросхемой начинают формировать ответ.

Предположим, что ответ (затребованный из оперативной памяти код) стабилизируется на шине данных только к моменту t_v в третьем такте цикла, то есть цикл памяти длится от t_r до t_v. Следовательно, в течение второго такта все устройства, участвующие в обмене, должны ждать получения ответа.

Контроллер памяти «знает» временные характеристики микросхем памяти. Чтобы процессор напрасно не ждал ответа и мог заняться выполнением других действий,

контроллер формирует на шине управляющий сигнал WAIT# (от wait — ожидание), занимающий столько тактов, сколько должен ждать ответа процессор. В ситуации, изображенной на рис. 10.1, сигнал WAIT# длится один такт.

В момент времени t_d биты передаваемого кода появляются на шине, и после того как к моменту t_v закончится период их стабилизации, процессор переводит управляющий сигнал MEMR# в неактивное состояние. После окончания этого такта шина готова к выполнению нового цикла чтения или записи. Цикл записи не имеет принципиальных отличий от цикла чтения и потому здесь не рассматривается.

Обсуждаемый цикл шины занимает три такта. Если микросхема памяти обладает меньшим быстродействием и цикл памяти длится дольше, то цикл шины может занять четыре и более тактов процессора. Если микросхема памяти работает быстрее и цикл памяти укладывается в один такт, то цикл шины может занять всего два такта, так как вводить такт пропуска с помощью сигнала WAIT# не нужно. Еще более быстрые микросхемы памяти при рассмотренной структуре цикла шины не приведут к дальнейшему улучшению эффективности, так как цикл с такой структурой в любом случае не может занимать меньше двух тактов.

Преимуществом синхронных шин является более простая и дешевая их реализация. Поэтому они распространены довольно широко. Вместе с тем у синхронных шин имеется ряд недостатков. К ним относится снижение общей эффективности передач, вызванное тем, что цикл шины может занимать только целое количество тактов. Пусть, например, процессор и микросхема памяти способны закончить обмен за 2,1 такта, но так как цикл шины может занимать только целое количество тактов, он займет три такта. Следовательно, процессор и память потеряют 0,9 такта на вынужденное ожидание при выполнении каждой операции обмена, то есть общее снижение эффективности за счет этого простоя составляет примерно 43 %. Следует также упомянуть вынужденную подстройку быстродействия шины к скорости самого медленного из подсоединенных к ней передающих устройств.

10.1.2. Цикл чтения асинхронных шин

Для того чтобы избавиться от отмеченных недостатков синхронных шин, были разработаны асинхронные шины, у которых привязка к тактовым синхроимпульсам отсутствует. Любое действие выполняется столько времени, сколько ему фактически требуется. Один и тот же цикл шины для различных пар устройств может иметь различную длительность.

Рассмотрим изображенную на рис. 10.2 упрощенную временную диаграмму цикла чтения для асинхронной шины. Кроме использованного ранее сигнала MEMR#, асинхронной шине требуется еще два управляющих сигнала — BEG#, роль которого сводится к оповещению «заинтересованного» устройства о том, что оно может начать свое действие, и сигнал FIN#, который отправляется устройством

после завершения его части работы. Эти два сигнала в цикле шины играют своеобразную синхронизирующую роль.

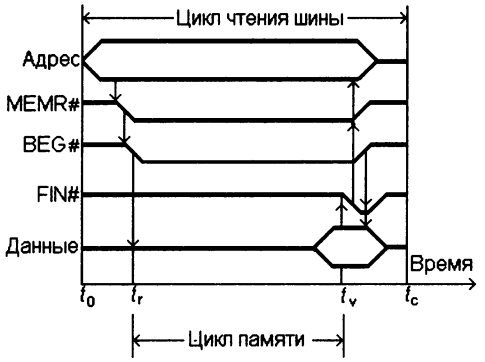


Рис. 10.2. Цикл чтения асинхронной шины

Для пояснения причинно-следственных связей в цикле шины на временной диаграмме асинхронной шины используются стрелки, начинающиеся от графика сигнала (адреса, данных), который вызывает действие, и заканчивающиеся у графика устройства, которое это действие исполняет.

Пусть, как и ранее, в момент времени t_0 процессор выставляет на шину адрес требуемого байта. Когда он стабилизируется, процессор формирует нулевое значение управляющего сигнала MEMR# — на рисунке для отражения этой связки поясняющая стрелка идет от графика **Адрес** к графику MEMR# (далее пояснения по поводу стрелок отсутствуют).

После стабилизации указанного сигнала формируется сигнал BEG#, извещающий оперативную память о том, что она может начинать чтение запрошенного байта. Фактически цикл памяти начинается в момент t_r, когда сигнал BEG# стабилизируется.

После стабилизации кода на шине данных и окончания цикла контроллер памяти устанавливает сигнал FIN#, извещая тем самым об окончании своей части действий. Стабилизация сигнала FIN# вызывает отключение сигналов BEG# и MEMR#, а также освобождение адресных линий. Чтобы привести систему в исходное состояние, позволяющее начать новый цикл, нужно убрать сигнал FIN#. Для этого используется фаза отключения сигнала BEG#, которая одновременно вызывает освобождение линий данных, что означает завершение цикла шины в момент t_c.

Цепочка причинно-следственных связей цикла асинхронной шины в краткой форме выглядит следующим образом.

1. Стабилизация адреса вызывает формирование сигнала MEMR#.
2. Стабилизация сигнала MEMR# вызывает формирование сигнала BEG#.
3. Стабилизация сигнала BEG# запускает цикл памяти.

4. Стабилизация данных (завершение цикла памяти) вызывает формирование сигнала FIN#.
5. Стабилизация сигнала FIN# вызывает отключение сигналов BEG#, MEMR#, а также адресных линий.
6. Отключение сигнала BEG# вызывает отключение сигнала FIN# и освобождение линий данных.

Такая последовательность взаимосвязанных сигналов называется **полным квитированием** (от итал. quitanza — квитанция[1]), так как каждое участвующее в обмене устройство как бы посылает и принимает *квитанцию*, заставляющую начать действие, или оповещающую о его завершении.

Асинхронная организация цикла шины не содержит связанных с ожиданием пауз, требует минимальных временны́х издержек, а также обеспечивает каждому устройству такую длительность цикла шины, которая соответствует его возможностям, независимо от длительностей циклов для других устройств. Платой за повышение эффективности является большая стоимость асинхронных шин по сравнению с синхронными.

10.1.3. Блочные циклы шины

Количество битов, которые передаются по шине за один рассмотренный ранее цикл чтения или записи, равно разрядности шины данных. Это могут быть одиночные байты, слова, двойные слова и т. д. Однако, как было выяснено при обсуждении кэша, данные из оперативной памяти желательно передавать пакетами (блоками), длина которых совпадает с длиной строки кэша. Понятно, что передача пакетами эффективнее передачи одиночными байтами или словами, так как можно сэкономить время на получение доступа к шине во время арбитража (см. 4.1.3), на выставление адреса и т. д.

Блочные (пакетные) циклы шины аналогичны по организации и получаемым преимуществам режиму BEDO микросхем памяти (см. 7.4.4). Для организации пакетного режима контроллеру памяти передается количество байтов (слов), из которых должен состоять блок. Далее без освобождения шины из памяти передается по одному байту (слову) в течение каждого такта до полного формирования пакета. Таким образом, при передаче блока, состоящего, например, из 4 байтов (слов), требуется не 12 тактов, а всего 6.

10.1.4. Циклы без освобождения шины

Обычно после выполнения одиночной или блочной передачи участвовавшие в обмене устройства освобождают шину и уступают ее другим устройствам компьютера, нуждающимся в обмене.

---

[1] Квитанция — официальный документ, расписка установленной формы в принятии денег, ценностей и т. д.

Однако в многопроцессорных системах, в которых процессоры параллельно выполняют различные части одной и той же программы, необходимо гарантировать, чтобы два различных процессора не получили одновременный доступ к значению одной и той же величины с целью ее модификации.

Допустим, в программе используется некоторая переменная *A*, значение которой управляет выбором ветви в условном операторе. Различные процессоры, выполняя различные подпрограммы, могут изменять значение этой переменной, чтобы выбрать нужную ветвь. Пусть, например, в некоторый момент времени первый процессор прочитает значение этой переменной и освободит шину, чтобы на основании полученного значения переменной *A* вычислить ее новое значение, принять решение о выборе следующего варианта, а затем записать новое значение в память. Предположим далее, что в промежуток времени между чтением и записью нового значения переменной *A* шину захватывает второй процессор и также считывает значение переменной *A* с той же самой целью — выбрать на основании считанного значения новую ветвь для своего участка вычислений. Следовательно, выбор варианта второй процессор будет делать, основываясь на уже устаревшем значении управляющей переменной *A*, так как к этому времени первый процессор может изменить значения многих величин, участвующих в вычислении значения *A*. В таких случаях важно организовать работу с шиной так, чтобы она оставалась в монопольном владении одного из процессоров до полного завершения работы со значением такой переменной (то есть на период считывания, изменения значения и записи нового значения). Для этого используются **циклы без освобождения шины**.

10.2. Конвейерный режим шины

Широкое использование конвейеризации в микросхемах памяти, о которой шла речь ранее, привело к существенному увеличению их эффективности. Этот же подход позволяет повысить производительность и шин компьютера. На рис. 10.3 изображена упрощенная схема последовательных тактов работы шины, использующей конвейеризацию.

Чтобы организовать конвейер, необходимо выделить в работе устройства несколько самостоятельных этапов и поручить выполнение каждого из них отдельному узлу, совместив их работу во времени. В рассматриваемой упрощенной ситуации цикл шины разбит на четыре этапа: фаза арбитража (А), фаза запроса (З), фаза проверки и выявления ошибки в запросе (О), фаза передачи запрошенных данных (Д).

Во время фазы арбитража арбитр по принятому алгоритму выбирает устройство, получающее доступ к шине. Фаза запроса включает формирование адреса и необходимых управляющих сигналов. На следующей фазе проверяется наличие ошибок в запросе, и при их отсутствии цикл шины содержит последнюю фазу, связанную с передачей запрошенных данных. В изображенной на рис. 10.3 ситуации предполагается, что запросы на выполнение операции чтения/записи не содержат ошибок.

Рис. 10.3. Упрощенная схема конвейерного режима шины

Рассмотрим прохождение по шине нескольких последовательных операций чтения/записи, каждой из которых соответствует отдельный цикл действий на конвейере. Для удобства изложения такой цикл будем называть **транзакцией**.

- **Первый такт.** Первая транзакция проходит фазу арбитража.
- **Второй такт.** Первая транзакция проходит фазу запроса, а вторая транзакция — фазу арбитража.
- **Третий такт.** Первая транзакция вышла на фазу проверки ошибки, вторая — на фазу запроса, в это же время третья транзакция проходит фазу арбитража.
- **Четвертый такт.** Первая транзакция завершается фазой передачи данных, вторая транзакция проходит этап проверки ошибок, третья проходит фазу запроса, а арбитраж проходит уже четвертая транзакция.
- **Пятый такт.** Вторая транзакция вышла на завершающую фазу передачи данных. Третья проходит проверку ошибок, четвертая вступила в фазу запроса, а пятая проходит арбитраж.
- **Шестой такт.** Третья транзакция завершается передачей данных. Четвертая проходит проверку ошибок, пятая находится на фазе запроса, а шестая — на фазе арбитража.
- **Седьмой такт** и т. д.

Как видно из протокола прохождения конвейера, каждая транзакция по отдельности длится столько же времени, сколько занимает неконвейерный цикл шины. Зато общая пропуская способность шины, подсчитанная за включающий несколько транзакций промежуток времени, оказывается существенно выше. Увеличение производительности может достичь четырехкратного уровня за счет того, что в установившемся режиме на конвейере шины одновременно выполняются четыре различные фазы четырех разных транзакций.

Следует обратить внимание на то, что на начальных этапах одновременно выполняется меньшее количество фаз цикла: на первом этапе — только одна, на втором — две, на третьем — три. И только на четвертом этапе конвейер работает

с полной нагрузкой. Этот период называется **заполнением конвейера**, он неизбежен в работе любого конвейера. Кроме периода заполнения конвейера существует еще и симметричный ему период **освобождения конвейера**, связанный с завершением запрошенных операций. Наличие периодов заполнения и освобождения, естественно, снижает эффективность конвейера.

10.3. Многошинная архитектура

Несмотря на возросшие с течением времени разрядности и тактовые частоты, несмотря на увеличение производительности шин, достигнутое с помощью конвейеризации и пакетного режима, одна общая шина в составе компьютера оказалась не способной обеспечить все необходимые передачи кодов и сигналов. Поэтому довольно быстро в архитектуре компьютеров появились дополнительные специализированные шины, которые связывают между собой различные группы устройств с различными скоростями обмена и различными требованиями к производительности. Архитектура компьютеров, в составе которых имеется более одной шины, называется **многошинной**.

Вначале в компьютерах была выделена шина, обеспечивающая обмен с низкоскоростными внешними устройствами, такими как клавиатура, мышь и принтер. Затем появилась еще одна шина с более высокой производительностью, к которой оказалось удобно подсоединять дисковые накопители. Относительно недавно выделилась шина для передачи на дисплей — графическая шина (шина графического адаптера) и т. д.

Когда в состав компьютеров стали включать внешний кэш, он подсоединялся к системной шине, связывающей процессор и оперативную память (см. рис. 9.1). Эта архитектура оказалась неэффективной, потому что кэш простаивал из-за низкой скорости шины. В 1997 г. была предложена архитектура **двойной независимой шины**, или **архитектура DIB** (от Dual Independent Bus), подразумевавшая наличие двух отдельных шин, одна из которых — **шина кэша**, или **шина BSB** (от Back Side Bus), — связывает процессор с внешним кэшем, а вторая — **системная шина**, или **шина FSB** (от Front Side Bus), — связывает процессор с оперативной памятью. Наличие двух независимых шин, по которым процессор может получить доступ к данным, передающимся по любой из шин одновременно и параллельно, более чем в три раза ускоряет работу внешнего кэша.

Итак, современные персональные компьютеры обычно содержат (рис. 10.4):

- шину низкоскоростных внешних устройств (клавиатура, мышь, принтер);
- шину высокоскоростных внешних устройств (магнитные, оптические диски);
- шину графического адаптера, для передачи высококачественных цветных движущихся изображений на экран дисплея;
- системную шину (шину памяти), связывающую процессор и оперативную память;
- шину кэша, связывающую процессор и внешний кэш.

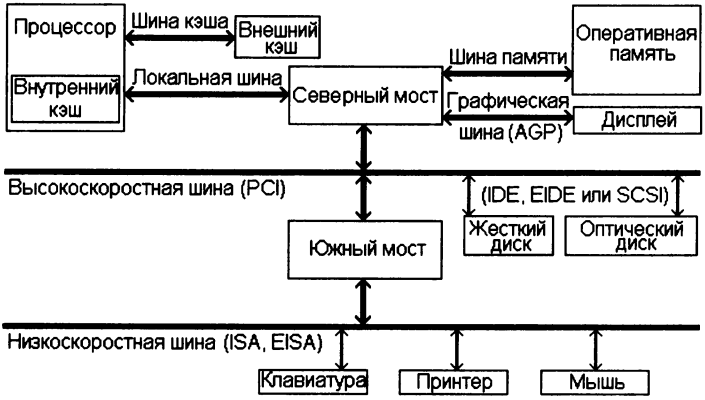


Рис. 10.4. Многошинная архитектура персонального компьютера

10.3.1. Основные типы шин

В процессе развития архитектуры и технологии производства было создано несколько типов шин. Наиболее старыми, но до сих пор используемыми являются шины **ISA** (от Industry Standard Architecture — стандартная промышленная архитектура) и **EISA** (от Extended ISA — расширенная ISA). Это традиционные, дешевые и универсальные шины, служащие для подключения медленных внешних устройств, например таких как мышь и клавиатура, с тактовой частотой до 8 МГц и малой скоростью обмена — от 8 до 33 Мбайт/с. Шины ISA имели разрядности 8 и 16 бит, а шины EISA — 32 бита.

Позднее появилась шина **PCI** (от Peripheral Component Interconnect — соединитель периферийных компонентов) — высокоскоростная шина для подсоединения дисковых и графических устройств со скоростью обмена до 500 Мбайт/с и тактовой частотой 66 МГц. Разрядности шин PCI — 64 и 128 бит.

С 2004 г. в качестве перспективного стандарта, который должен прийти на смену шине PCI, рассматривается шина **PCI Express**. Для нее введена единица скорости передачи данных, равная 256 Мбайт/с. Уже выпущены шины этого стандарта, имеющие скорость передачи 16x, то есть 4 Гбайт/с.

Шины **IDE** (от Integrated Drive Electronics — интегрированные электронные устройства) и **EIDE** (от Extended IDE — расширенная IDE). Это шины и одновременно стандарт (интерфейс) для подключения жестких, оптических дисков и мобильных дисководов со скоростью обмена до 20 Мбайт/с.

Шина **SCSI** (от Small Computer System Interface — интерфейс малых вычислительных систем). Это шина и стандарт, которые предназначены для подключения высокопроизводительных дисковых устройств со скоростью обмена до 80 Мбайт/с. Отметим, что шины SCSI дороже шин EIDE.

Шина **AGP** (от Advanced Graphic Port — улучшенный графический порт). Это шина для подключения видеоплат со скоростью обмена от 256 Мбайт/с до 1,06 Гбайт/с. При этом скорость обмена 256 Мбайт/с считается условной едини-

цей измерения. Например, скорость обмена, равную 528 Мбайт/с, принято обозначать AGP 2x, а 1,06 Гбайт/с — AGP 4x.

Шина **USB** (от Universal Serial Bus — универсальная последовательная шина). Допускается одновременное подключение к такой шине до 127 внешних устройств (принтеров, сканеров, переносных запоминающих устройств и т. д.). Скорость обмена данными у шины доходит до 12 Мбайт/с, а у ее модификации USB 2.0 — до 60 Мбайт/с.

Шина **FireWire** (от fire wire — огненный провод), или **IEEE 1394**. Это шина и стандарт, предназначенные для подключения высокоскоростных внешних устройств типа цифровых фото- и видеокамер. Обеспечивается скорость обмена до 50 Мбайт/с.

10.3.2. Синхронизация и шины

Находящийся на материнской плате основной тактовый генератор компьютера вырабатывает высокостабильные импульсы, которые используются для синхронизации работы процессора, микросхем памяти, системной и всех остальных шин, а также разнообразных внешних устройств. В связи с этим частота импульсов, вырабатываемых тактовым генератором, называется **опорной**, или **базовой**, частотой. В литературе такую частоту иногда называют **Host Bus Clock** (дословно — часы хозяйской шины, подразумевается частота системной шины). Считается, что первой стандартной базовой частотой у персональных компьютеров была частота 4,77 МГц. В настоящее время базовые частоты увеличились до 800–1200 МГц.

В первых моделях компьютеров IBM PC системная шина, процессор и микросхемы памяти работали на одной и той же опорной частоте тактового генератора. Это было неэффективно, так как более скоростные устройства должны были приспосабливаться к скорости самого медленного устройства.

В связи с широким внедрением параллелизма в архитектуру компьютеров оказалось более удобным и более производительным допустить работу различных устройств с различной тактовой частотой. А чтобы не вводить несколько тактовых генераторов, которые работают на разных частотах, но при этом занимают много лишнего места и к тому же нуждаются в дополнительной взаимной синхронизации, применили масштабирование одной и той же базовой частоты от одного тактового генератора. Для обладающих высокой скоростью работы микросхем процессоров применяется внутреннее умножение базовой частоты на некоторый числовой коэффициент, например на 2, 4, 6 и т. д. Эту частоту часто обозначают **CPU Clock** (частота центрального процессора) или **Core Speed** (скорость ядра).

Тактовая частота микросхем памяти не вводится, поскольку имеется довольно сложная зависимость времени получения или записи данных от времени доступа микросхемы и ее цикла памяти. Именно поэтому для характеристики микросхем памяти вводятся различные величины — быстродействие, время задержки, цикл памяти и скорость обмена. Но при этом шина памяти FSB всегда имеет частоту,

однозначно определяемую частотой тактового генератора, которая служит также для характеристики возможностей микросхем памяти.

Для более медленных шин устройств ввода/вывода применяется деление опорной частоты на соответствующий возможностям их шин и самих устройств коэффициент. При этом получают, например, для шин ISA частоту 8, 16 МГц и т. д. (она называется **ISA Bus Clock**), а для шин PCI частоту **PCI Bus Clock**, равную 66, 260 МГц и т. д.

10.3.3. Чипсет

Для согласования операций обмена и управления в сложной иерархической системе шин в персональных компьютерах служит специальный набор микросхем компьютера, который принято называть **чипсетом** (от chip set — набор чипов).

ВНИМАНИЕ

Чипсет представляет собой базовый набор специализированных микросхем, при подключении которых друг к другу формируется функциональный блок компьютера, обеспечивающий связь между основными компонентами компьютера — процессором, памятью, шинами, периферийными устройствами и т. д.

Чипсет фактически является связующим звеном между всеми компонентами материнской платы. Именно он определяет возможности модернизации компьютера путем замены отдельных устройств более эффективными.

Микросхемы чипсета содержат в себе контроллеры прерываний, контроллеры прямого доступа к памяти, контроллеры шин и т. д. В одну из микросхем набора обычно входят также часы реального времени со CMOS-памятью, а иногда и клавиатурный контроллер, однако эти блоки могут присутствовать в компьютере и в виде отдельных микросхем. В последних разработках в состав микросхем чипсета стали включаться контроллеры внешних устройств.

В «обязанности» чипсета, в частности, входит:

- обслуживание управляющих сигналов процессора;
- формирование управляющих сигналов для внешнего кэша;
- формирование адреса и управляющих сигналов для микросхем оперативной памяти;
- обеспечение когерентности данных для всех уровней кэша и оперативной памяти;
- обеспечение взаимосвязи между всеми типами шин компьютера ;
- арбитраж контроллеров шин и т. д.

Характеристики чипсета определяют большинство важнейших параметров компьютера, в том числе:

- возможные типы и частоты центрального процессора;
- скоростные характеристики внешнего кэша и его допустимый объем;

- типы, объемы и максимальное количество модулей динамической памяти;
- возможные частоты шин и т. д.

Тип используемого чипсета существенно влияет на производительность компьютера в целом. При одинаковых базовых компонентах компьютера — процессоре, оперативной памяти, кэше, графическом контроллере (контроллере дисплея) и жестком диске — производительность машин, собранных на разных чипсетах, может различаться на 30 % и более.

Отметим, что реализация такого функционально сложного блока с помощью только одной микросхемы оказывается по ряду причин невыгодной.

В настоящее время чипсеты персональных компьютеров функционально состоят из микросхем **северного моста**, отвечающих за соединение между собой процессора, оперативной памяти и дисплея, и микросхем **южного моста**, которые отвечают за работу с остальными внешними устройствами (дисками, клавиатурой и т. д.).

В связи с появлением чипсетов в состав компьютеров вместо одной системной шины включили **локальную шину**, которая служит для соединения процессора с чипсетом северного моста, и **шину памяти**, соединяющую с этим чипсетом оперативную память (рис. 10.4).

Дополнительный материал по обсуждавшимся в главе вопросам можно найти в изданиях [18], [30], [34], [35].

Контрольные вопросы и упражнения

1. Дайте определения техническим характеристикам шины.
2. Опишите принцип работы мультиплексных шин.
3. В чем состоит перекос шины?
4. Что понимается под циклом шины? Какие циклы шины вам известны?
5. Изобразите временную диаграмму цикла синхронной шины и опишите связанную с ним последовательность действий.
6. Почему в цикле синхронной шины возникают такты ожидания?
7. Изобразите временную диаграмму цикла асинхронной шины и опишите связанную с ним последовательность действий.
8. Что понимается под полным квитированием?
9. Дайте сравнительную характеристику синхронных и асинхронных шин.
10. Охарактеризуйте блочные циклы шин.
11. Для чего нужны циклы без освобождения шины?
12. Опишите конвейерный режим работы шин.
13. В чем преимущество многошинной архитектуры компьютера?

14. Опишите архитектуру с двойной независимой шиной.
15. Поясните смысл терминов «шина кэша», «системная шина», «шина FSB», «шина BSB», «архитектура DIP».
16. Назовите основные типы шин, используемых в компьютерах, и дайте их сравнительные характеристики.
17. Как формируются рабочие частоты для различных устройств компьютера?
18. Для чего нужен чипсет? Что он определяет? Охарактеризуйте его функции.
19. Поясните смысл терминов «северный мост», «южный мост», «локальная шина», «шина памяти».

Глава 11
Улучшение эффективности процессора

Рассмотренные ранее подходы к реализации памяти, кэша и шин компьютера существенно влияют на его эффективность. Однако самое большое влияние на производительность компьютера оказывают характеристики процессора, которые также могут быть существенно улучшены, и не только путем изменения его физических параметров. Основными техническими характеристиками процессоров в настоящее время считаются:

- архитектура системы команд процессора;
- модель процессора, связанная с его системой команд;
- внутренняя тактовая частота;
- разрядность или длина машинного слова;
- объем внутреннего кэша L1;
- производительность;
- плотность логических элементов;
- линейный размер логических элементов;
- форм-фактор, определяющий геометрические размеры, форму, количество контактов посадочного гнезда (сокета), а также способ подключения микросхемы процессора к материнской плате.

Архитектура системы команд и модели компьютеров рассматриваются в главах 15 и 16. Понятие быстродействия процессора или его производительности обсуждается в главе 14. А в данной главе описываются архитектурные подходы и решения, касающиеся логического устройства процессора.

11.1. Микроархитектура процессора

Уже на начальных этапах развития вычислительных систем стало ясно, что компьютер — это сложная система, которую следует рассматривать как *многоуровневую структуру* (рис. 11.1). На самом нижнем уровне этой структуры находятся *физические элементы*, из которых изготовлена машина: электромагнитные реле, электронные лампы накаливания, интегральные схемы, резисторы, конденсаторы, соединительные провода, шины, линии связи и т. д. На этом уровне обсуждаются физические характеристики устройств: геометрические размеры (высота, ширина, толщина), электрические параметры (уровни напряжения и тока в цепи, сопротивление, емкость, индуктивность), временны́е последовательности прохождения импульсов тока и т. д.

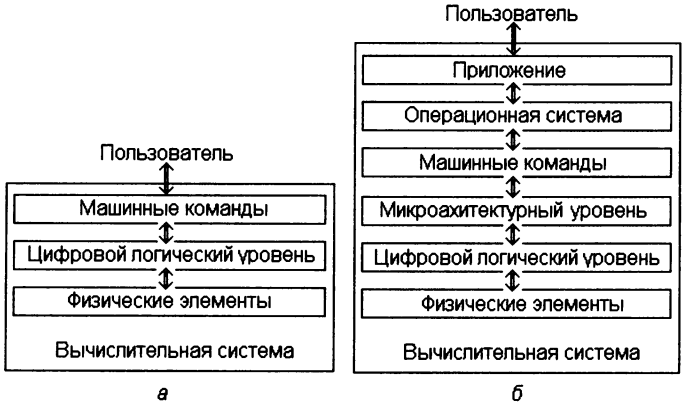


Рис. 11.1. Многоуровневая организация вычислительных систем

Следующий уровень вычислительной системы состоит из различных *вентилей*, которые реализуют все необходимые функции цифровой логики, из-за чего его называют *цифровым логическим* уровнем, или уровнем **цифровой схемотехники**. На этом уровне происходит абстрагирование от отдельных физических устройств и их характеристик. Обсуждение ведется в терминах двоичных цифр и связывающих их логических и арифметических операций. Этот уровень организации компьютера довольно подробно рассматривался в 3.2. Физический и цифровой логический уровень — это сфера деятельности разработчиков вычислительной техники, специалистов в области схемотехники.

Над указанными уровнями в машинах первого поколения располагался уровень машинных команд (рис. 11.1, *а*), которые подобны командам, рассматривавшимся в главе 4. Пользователь взаимодействовал с аппаратурой компьютера на языке машинных команд. При этом в его поле зрения отсутствовали отдельные вентили и поступающие в них сигналы. Рассмотрение велось на уровне битов, байтов, полей, кодов операций, адресов операндов, флажков процессора и т. д. Ясно, что для практического программирования хватает общего представления о первых двух уровнях, но зато необходимо детальное знание уровня машинных команд.

Довольно быстро специалисты обнаружили, что выполнение процессором большинства машинных команд включает в себя практически одинаковые действия. Как правило, в различных машинных командах по одной и той же схеме выполняется выборка операндов из регистров процессора, из полей памяти или из самой команды. Точно так же одинаково отправляются на хранение результаты выполнения команд, и по одним и тем же правилам формируются значения флажков. Кроме того, имеются тесно связанные команды, реализующие родственные действия, например: сложение, инвертирование знака, вычитание, инкремент, декремент. Было бы неразумно для каждой машинной команды предусматривать отдельные аппаратные схемы выполнения, в которых дублируются выполняемые по общим правилам действия.

В связи с этим еще в 1951 г. М. Уилкс из Кембриджского университета с целью упрощения аппаратного обеспечения компьютера предложил подход, который впоследствии получил название **микропрограммирования**. Для его реализации в структурную организацию вычислительной системы между цифровым логическим уровнем и уровнем машинных команд включается дополнительный **микроархитектурный** уровень (рис. 11.1, *б*).

На этом рисунке кроме микроархитектурного уровня изображены еще два дополнительных уровня в организации вычислительной системы, которые также появились позднее. Это уровень операционной системы, автоматизирующий доступ программам и пользователям ко всем ресурсам компьютера, а также уровень приложений, которые обычно обеспечивают пользователям значительно более удобный интерфейс, чем работа на уровне машинных команд или на уровне операционной системы.

Основу микроархитектурного уровня образуют **микрокоманды**. Каждая микрокоманда, выдавая управляющие сигналы на отдельные вентили аппаратных схем, определяет некоторое простейшее действие процессора, например задает выбор кода из конкретного регистра процессора, установку флажка в соответствующее результату значение и т. д. Поэтому выполнение любой машинной команды фактически представляет собой выполнение соответствующей последовательности микрокоманд. Например, команда `add ax,cx` из листинга 4.2 может быть реализована как последовательность микрокоманд, изложенная ниже.

1. Микрокоманда, выбирающая операнд из регистра `ax` и перемещающая его в арифметико-логическое устройство.
2. Микрокоманда, выбирающая операнд из регистра `cx` и перемещающая его в арифметико-логическое устройство.
3. Микрокоманда, формирующая новое значение указателя команд.
4. Микрокоманда, выполняющая сложение двух операндов.
5. Микрокоманда, отправляющая полученную сумму в регистр `ax`.
6. Микрокоманда, формирующая значения флажков `zf` и `sf`.

Если же, например, нужно выполнить команду `sub ax,cx`, то для ее реализации может быть использована почти такая же последовательность микрокоманд,

только после второй микрокоманды нужно включить микрокоманду, инвертирующую вычитаемое, и микрокоманду, выполняющую его инкремент. Две добавочные микрокоманды образуют дополнительный код вычитаемого и тем самым сводят вычитание к сложению с отрицательным числом.

Последовательности микрокоманд, аналогичные рассмотренной, могут быть построены для каждой машинной команды, принадлежащей системе команд процессора. Понятно, что такие последовательности существенным образом зависят от базового набора микрокоманд, включенных в микроархитектурный уровень. В связи с этим в различных реализациях микроархитектурного уровня формируемые последовательности микрокоманд могут быть разными.

Из приведенных примеров видно, что введение микроархитектурного уровня значительно упрощает разработку аппаратных средств, которые теперь должны уметь выполнять значительно меньше различных действий, чем при аппаратной реализации каждой машинной команды в отдельности.

Отметим еще одну важную особенность микрокоманд. Любая микрокоманда выполняется за один такт работы процессора, в то время как выполнение машинной команды может потребовать нескольких его тактов.

С точки зрения микроархитектурного уровня отдельная машинная команда — это целая программа. Она выполняется либо специальной программой, которую принято называть **микропрограммой**, либо аппаратными средствами процессора. Основными функциями микропрограммы являются построение для каждой машинной команды из программы пользователя соответствующей последовательности микрокоманд и организация их выполнения. Эти действия выполняются микропрограммой для каждой машинной команды по отдельности, то есть программа пользователя фактически интерпретируется микропрограммой, поэтому микропрограмма считается *интерпретатором* выполняющейся программы.

Из-за огромного влияния микропрограммы на эффективность работы компьютера в целом она разрабатывается особенно тщательно. Обычно микропрограмма имеет небольшие размеры и хранится в постоянном запоминающем устройстве компьютера в виде, недоступном для внесения изменений.

С течением времени границы между функциями микропрограммы и аппаратной реализацией микрокоманд меняются. Появление новых аппаратных возможностей приводит к переносу операций, закрепленных за программным обеспечением, на аппаратный уровень, их встраиванию в аппаратные средства. Это приводит к увеличению скорости выполнения машинных команд. С другой стороны, программная реализация машинной команды удешевляет разработку и производство компьютера. Конструкторам вычислительной техники приходится постоянно искать компромисс между противоречивыми требованиями, когда улучшение одних параметров приводит к ухудшению других. Сложность в изучении микроархитектурного уровня связана с тем, что разработчиками компьютеров предложено огромное количество различных вариантов его реализации. Подробное изучение микроархитектурного уровня выходит за рамки данного учебника.

11.2. Конвейерная архитектура процессора

Впервые параллелизм на уровне машинных команд в виде совмещения во времени различных этапов выполнения команды был реализован в архитектуре выпущенной в 1959–1961 гг. машины IBM Stretch. В ее конструкции был использован *буфер вызова команд с упреждением*. Очередная команда программы попадала из оперативной памяти в этот буфер заранее, еще до завершения предыдущей команды. В момент, когда очередь доходила до выполнения следующей команды, она выбиралась уже из буфера, что требовало гораздо меньше времени, чем выборка из оперативной памяти. Кроме того, в машине был организован *опережающий* просмотр находящихся в буфере команд для определения адресов и предварительной выборки операндов. К тому моменту, когда процессор заканчивал дешифрацию выбранной команды, ее операнды уже находились в арифметико-логическом устройстве. И сразу же в буфер переносилась следующая команда. В выпущенной в 1962 г. машине ATLAS этот подход был расширен до полноценной конвейерной архитектуры процессора.

На рис. 11.2 представлен порядок выполнения машинной команды процессором, не имеющим конвейера. По горизонтальной оси рисунка располагаются этапы выполнения команды процессором, а по вертикальной оси отложены такты его работы. Для упрощения обсуждения будем считать, что процессор осуществляет выполнение *любой* команды за пять этапов, а на выполнение каждого этапа требуется один такт. Тогда на выполнение одной команды процессору требуется пять тактов: на первом такте процессор выполняет выборку операнда, на втором — декодирование, на третьем — выборку операндов, на четвертом — исполняется действие команды, а на последнем пятом — записываются результаты. Точно по такой же схеме выполняется каждая следующая команда, причем процессор не начинает выполнение следующей команды, пока не завершит выполнение предыдущей. Поэтому промежуток времени между завершениями двух последовательных команд также равен пяти тактам (рис. 11.2).

Обратите внимание на то, что схема процессора, которая выполняет, например, выборку команды, простаивает в течение четырех тактов, пока процессор полностью не завершит выполнение всей команды. Аналогичным образом после выполнения своих «обязанностей» простаивают все остальные схемы процессора.

Идея организации конвейерной обработки состоит в том, чтобы с помощью выделения этих схем процессора в относительно самостоятельные блоки организовать их непрерывную работу — сразу после обработки своего этапа текущей команды блок должен переключаться на выполнение того же самого этапа у следующей команды.

При наличии конвейера в микросхемах памяти, в шинах и т. д. архитектура компьютера в целом не считается конвейерной. Только организация конвейерного выполнения машинных команд в процессоре позволяет считать архитектуру компьютера конвейерной.

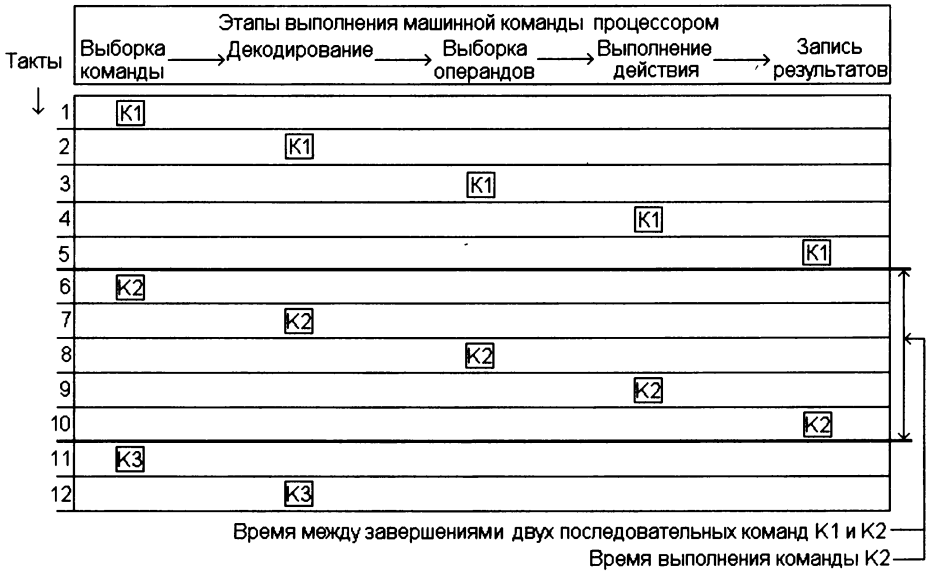


Рис. 11.2. Стандартное выполнение команд процессором

ВНИМАНИЕ

Группа блоков процессора, обеспечивающих одновременное выполнение различных этапов различных машинных команд, называется конвейером, блоки, из которых он состоит, называются ступенями конвейера, а количество ступеней — длиной конвейера. Архитектура компьютера, в конструкции процессора которого используется конвейер, называется конвейерной.

Рассмотренная схема деления на блоки — не единственно возможная. Встречаются схемы с различным количеством ступеней и с различным распределением действий между ними. Количество ступеней конвейера является его основной характеристикой и потому включается в его название: например, пятиступенчатый конвейер. Чтобы совмещение во времени разных этапов команды было более эффективным, их стараются выделять так, чтобы каждый этап занимал один такт работы процессора. Как выяснено в предыдущем разделе, таким свойством обладают *микрокоманды* процессора. Поэтому многие ступени конвейера реализуются как блоки выполнения микрокоманд.

На рис. 11.3 изображена ситуация, когда выполнение машинной команды осуществляется пятью блоками конвейера и процессор выполняет серию из восьми машинных команд, обозначенных K1, K2, ..., K8.

На первом такте команда K1 попадает в первый блок конвейера, который выполняет выборку команды из буфера команд, кэша или оперативной памяти в зависимости от ситуации.

Во время второго такта команда K1 проходит этап декодирования, а блок выборки в это же время занят обработкой команды K2.

Рис. 11.3. Пятиступенчатый конвейер выполнения команды

На третьем такте команда К1 поступает в блок выборки операндов, команда К2 дешифруется, а блок выборки занят командой К3.

Четвертый такт у команды К1 завешается получением результатов, у команды К2 — выборкой операндов, у команды К3 — дешифровкой, а у команды К4 — ее выборкой.

Во время пятого такта результаты выполнения К1 записываются в память или в регистр процессора, затем команда покидает конвейер. Команды К2–К4 передаются следующим блокам конвейера, а на конвейер поступает команда К5.

Заметим, что на первых четырех этапах происходит постепенное заполнение конвейера, его блоки последовательно подключаются к обработке машинных команд. На первом такте работает один блок, четыре следующих пока ожидают. На втором такте работают уже два блока и только три ожидают и т. д. Напомним, что такой период называется заполнением конвейера. Только после попадания команды К1 в пятый блок конвейер оказывается полностью загруженным — он переходит в установившийся режим, в котором одновременно работают все его блоки.

На шестом такте на конвейер поступает команда К6, а покидает его команда К2, затем поступает команда К7, а покидает К3. Последней (в рассматриваемом примере) на конвейер на восьмом такте попадает команда К8. Такты с пятого по восьмой установившегося режима конвейер работает с максимальной производительностью — все его блоки одновременно, параллельно выполняют свои этапы очередных команд.

По условиям рассматриваемого примера на восьмой команде серия машинных команд заканчивается. Поэтому на девятом и всех последующих тактах очередная команда на конвейер не поступает, а уже находящиеся на конвейере команды

продолжают переходить от блока к блоку, пока не покинут конвейер. Период освобождения рассматриваемого конвейера, так же как и период его заполнения, занимает четыре такта.

Понятно, что наибольший эффект от реализации конвейерного выполнения машинных команд достигается в установившемся режиме, когда все блоки конвейера работают одновременно. Сделаем примерный расчет повышения эффективности от реализации конвейерной обработки в рассматриваемом примере. Пусть, например, тактовая частота процессора равна 500 МГц, тогда один такт длится 2 нс. Если каждый этап выполняется за один такт, то одна машинная команда проходит пять ступеней конвейера за 10 нс. На первый взгляд, получается, что скорость обработки равна 100 млн. машинных команд в секунду и никакого преимущества от конвейеризации не получено. Но каждый этап команды, в том числе последний, выполняется за 2 нс, — следовательно, за 10 нс конвейер покидают 5 команд, и поэтому скорость обработки равна 500 млн. команд в секунду.

Будем считать, что производительность (пропускная способность) R процессора равна количеству выполненных им команд в единицу времени. В общем случае при подсчете производительности нужно исходить из количества команд, выполнение которых завершено процессором за единицу времени, то есть нужно использовать промежуток времени между двумя последовательными завершениями команд.

Пусть время такта равно t нс, а каждая команда выполняется за n тактов, тогда выполнение команды происходит за $T = nt$ нс. В процессоре, не имеющем конвейера, время выполнения одной машинной команды T совпадает с промежутком времени между двумя последовательными завершениями команд (см. рис. 11.2). Поэтому производительность процессора без конвейера равна $R = 1/T = 1/(nt)$.

Пусть теперь конвейер содержит n ступеней и каждая ступень проходится за один такт. Время выполнения одной команды по-прежнему равно $T = nt$ нс, но при использовании конвейерной организации, когда одновременно выполняются несколько машинных команд, количество команд, покидающих конвейер за единицу времени, больше, чем без конвейера. Команды покидают последний этап на каждом такте, поэтому промежуток времени между двумя последовательными завершениями команд равен t (см. рис. 11.3). Следовательно, в идеальном случае производительность конвейера равна $R = 1/t$ команд в секунду, то есть в n раз больше, чем без конвейера. Выполненные оценки не учитывают наличия периодов заполнения и освобождения конвейера, которые, очевидно, снижают производительность процессора.

Так как конвейерная архитектура на практике показала свою высокую эффективность, появились различные варианты ее дальнейшего развития. В частности, в процессор стали включать несколько конвейеров, которые могут функционировать одновременно. На рис. 11.4 приведен пример двухконвейерной архитектуры процессора. Общий для двух конвейеров блок выборки извлекает из буфера (из кэша или из оперативной памяти) сразу две команды и помещает каждую

из них на отдельный конвейер. Чтобы не задерживать остальные блоки конвейеров, пропускная способность блока выборки должна обеспечивать выполнение им своих «обязанностей» с соответствующей скоростью.

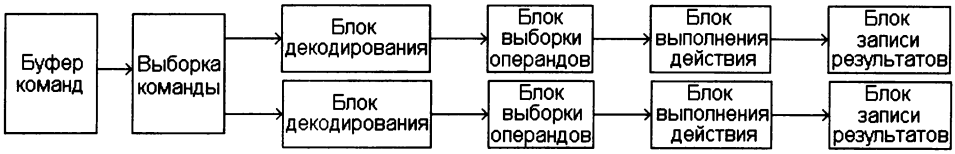


Рис. 11.4. Архитектура процессора с двумя конвейерами

ВНИМАНИЕ

Архитектура компьютера, процессор которого содержит несколько конвейеров выполнения машинных команд, называется суперконвейерной.

Находящиеся на разных конвейерах команды не должны конфликтовать друг с другом за одни и те же ресурсы, например, использовать одни те же регистры процессора, одни и те же адреса памяти, операнды и т. д. Кроме того, они не должны зависеть друг от друга, то есть результаты выполнения одной из них не должны использоваться как операнды другой. За этим должны следить специальные аппаратные схемы или программное обеспечение, например, специализированные трансляторы, учитывающие особенности архитектуры компьютера.

Из-за необходимости выполнения дополнительных проверок процессоры компьютеров различных моделей содержат только два-три конвейера. Дальнейшее увеличение количества конвейеров вызывает непропорциональное усложнение аппаратных схем и программных средств, разрешающих конфликты и распознающих возможности параллельного выполнения команд программы. Кроме того, из-за значительных временных затрат на эти проверки эффект от организации большого количества конвейеров, фактически, не приводит к увеличению производительности процессора.

11.3. Суперскалярная архитектура процессора

Анализ работы конвейера выявил, что самым критическим с точки зрения временных затрат является этап выполнения заданного в команде действия (сложения, умножения и т. д.). Остальные блоки конвейера зачастую вынуждены простаивать в ожидании его завершения. Для устранения простоев конвейера в это «узкое» место включили несколько **исполнительных** блоков (арифметико-логических устройств, сопроцессоров и т. д.), которые могут одновременно выполнять закодированные в машинных командах действия. Такой подход был назван **суперскалярной архитектурой**.

ВНИМАНИЕ

Архитектура, в которой используется большое количество параллельно работающих на этапе исполнения действия команды блоков, называется суперскалярной.

Схема суперскалярного процессора приведена на рис. 11.5. Блок выборки операндов может передавать операнды исполнительному блоку значительно быстрее, чем последний может их принимать, поэтому в суперскалярной архитектуре блок выборки «обслуживает» сразу несколько исполнительных блоков. Пока первый исполнительный блок занят своими «обязанностями», блок выборки успевает по очереди передать нужные операнды всем остальным блокам этого этапа и подготовить для первого блока следующие операнды.

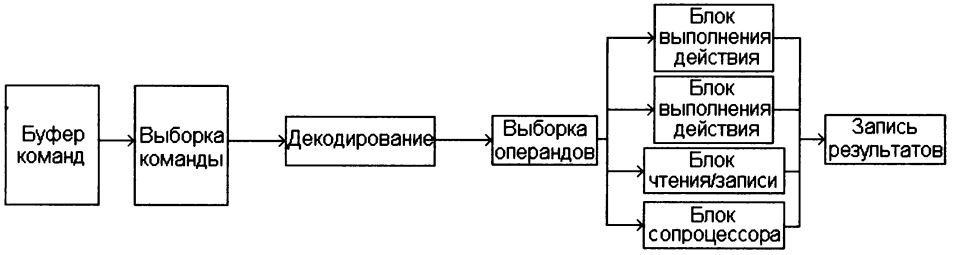


Рис. 11.5. Суперскалярная архитектура процессора

В дальнейшем появились процессоры, которые объединяют оба подхода к архитектуре процессора. Такие процессоры считаются суперскалярными суперконвейерными.

11.4. Динамическое исполнение машинных команд

Организация конвейерного исполнения машинных команд поставила ряд проблем, связанных с эффективностью функционирования конвейера. Многие из них сгруппировались вокруг периодов заполнения и освобождения конвейера, так как в установившемся режиме конвейер работает с максимальной эффективностью. Переход в установившийся режим возможен только во время выполнения линейных участков программ. Анализ структуры типичных программ показывает, что линейные участки состоят в среднем из 8–10 машинных команд и что программы содержат около 20 % команд перехода. Обрыв одного и начало нового линейного участка командой перехода, в которой организуется ветвление или цикл, приводят к необходимости освобождения и последующего заполнения конвейера. Большое количество таких периодов сводит на нет все преимущества конвейерной организации процессора.

Существуют и другие проблемы, снижающие эффективность конвейера. В частности, это проблемы, связанные с невозможностью выполнения команды, которой

нужны результаты выполнения другой, параллельно выполняемой команды. В этом случае приходится включать в работу конвейера такты ожидания или же применять другие искусственные приемы.

Для ослабления влияния обсуждаемых эффектов был разработан подход, который принято называть **динамическим исполнением**. Он представляет собой комбинацию трех технологий, обеспечивающих эффективную работу конвейера. Это **предсказание ветвления**, анализ потока данных и последующее **изменение порядка выполнения команд**, а также **спекулятивное исполнение**. Динамическое исполнение обеспечивает более эффективную работу процессора, в частности, за счет применения специальных методов построения трансляторов и операционных систем.

11.4.1. Изменение последовательности выполнения команд

Как только что было отмечено, проблемы с заполнением конвейера возникают не только при встрече с командой перехода — и на линейных участках программы также могут встретиться ситуации, вызывающие простои конвейера. Обсудим эти ситуации и способы разрешения проблем.

В стандартных архитектурах процессора все команды программы выполняются в том порядке, в котором они вызываются из памяти, то есть в естественном порядке, определяемом программой. Простои конвейера при такой последовательном выполнении команд программы вызываются *взаимозависимостями* между соседними командами. Различают три типа взаимозависимостей:

- **RAW** (от Read After Write — чтение после записи) — до завершения записи операнда нельзя начинать его чтение;
- **WAW** (от Write After Write — запись после записи) — до завершения записи операнда нельзя начинать следующую запись;
- **WAR** (от Write After Read — запись после чтения) — до завершения считывания операнда нельзя начинать запись его нового значения.

Рассмотрим, например, следующий фрагмент программы:

```
mov ax,2
mov bx,ax
```

В первой команде осуществляется запись в регистр `ax` кода числа 2, а в следующей команде код из этого регистра считывается для пересылки в регистр `bx`. Очевидно, что до завершения записи в регистр `ax` нового кода (точнее, до стабилизации нового содержимого регистра) пытаться что-либо читать из него бессмысленно. Аналогичным образом выглядят и остальные взаимозависимости.

Кроме того, соблюдение последовательного порядка выполнения команд требует, чтобы они завершались также в порядке, соответствующем программному. Требование завершения выполнения команд в программном порядке определяется необходимостью сохранения точного состояния процессора в момент возможного прерывания.

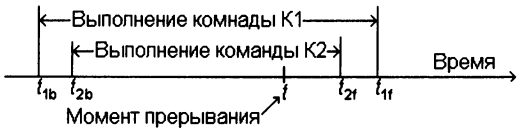


Рис. 11.6. К обоснованию требования программного порядка завершения команд

На рис. 11.6 показана ситуация, в которой нарушен порядок завершения команд. Пусть выполнение команды K1 началось в момент t_{1b}, а спустя некоторое время в момент t_{2b} началось выполнение команды K2. Допустим, что выполнение команды K2 занимает меньше тактов, чем выполнение команды K1, и команда K2 завершается в момент t_{2f}, а команда K1 — в момент t_{1f}, $t_{2f} < t_{1f}$.

Пусть в некоторый момент времени $t < t_{2f} < t_{1f}$ происходит прерывание, которое не связано с обсуждаемыми командами. В соответствии с порядком обработки прерывания вначале завершается выполнение команды K2, затем фиксируется содержимое всех регистров процессора, в том числе и содержимое регистра `ip`, который в момент прерывания содержит указатель на команду, следующую за K2. При стандартной трактовке прерывания (без наличия конвейера) считается, что все команды до той, на которую показывает указатель из `ip`, уже выполнены. Следовательно, начинать выполнение программы после обработки прерывания нужно с команды, следующей за K2. Но на рис. 11.6 видно, что к моменту t_{2f} команда K1 еще не завершена и результаты ее выполнения в результате прерывания окажутся потерянными. То есть при несоблюдении порядка завершения команд возможна ситуация, при которой прерывание внесет ошибку в выполнение программы. Чтобы такие ошибки не возникали в процессе выполнения команд на конвейере, приходится включать такты ожидания завершения предыдущих команд.

Для демонстрации возникновения простоев конвейера из-за наличия взаимозависимостей и необходимости ожидать момента завершения предшествующих команд рассмотрим простой пример. Допустим, что программа выполняется на процессоре с трехадресной системой команд, в котором имеются регистры общего назначения R1, R2, ..., R8. Действия команд будем записывать в паскалеподобном стиле, например, запись R3:=R0×R1 соответствует команде умножения содержимого регистров R0 и R1 с записью результата в регистр R3.

В табл. 11.1 показан последовательный порядок выполнения машинных команд программы на конвейере, в котором взаимозависимости команд учитываются включением необходимого количества тактов ожидания. В первом столбце указан номер такта работы процессора, во втором — номер команды программы, а третий столбец содержит описание заданного командой действия. Столбец «Поступление» содержит номер команды, которая на текущем такте поступает на конвейер. В столбце «Длительность» показано время выполнения команды в тактах работы процессора. Столбец «Завершение» содержит номер команды, выполнение которой на текущем такте завершается.

После поступления первой команды на конвейер в течение шести тактов (длительность первой команды) ни одна команда, которая считывает содержимое регистра R3 или записывает новые коды в R0, R1, R3, не может поступить на конвейер.

Таблица 11.1. Последовательное выполнение машинных команд на конвейере

Такт	Номер команды	Действие	Поступление	Длительность	Завершение
1	1	R3:=R0×R1	1	6	
2	2	R4:=R0+R2	2	3	
3	3	R5:=R0+R1	3	3	
4	4	R6:=R1+R3*			
5					
6					
7			4	3	1
8	5	R7:=R0×R8	5	6	2
9	6	R1:=R0×R0	6	6	3
10	7	R3:=R1*−R2			4

Поскольку вторая команда выполняет чтение из регистра R0, а регистры R1 и R3 вообще не использует, она не имеет взаимозависимости с первой и поступает на конвейер на втором такте.

Аналогичным образом на конвейер попадает третья команда, в которой из регистров R0 и R1 выполняется чтение, а остальные регистры, использованные в первой и второй командах, не затрагиваются вообще.

А вот четвертая команда имеет RAW-зависимость от первой. Ей требуется операнд из регистра R3 (в таблице отмечен звездочкой), который занят для записи в первой команде. Поэтому четвертая команда на конвейер не поступает. Все команды, следующие за четвертой, также переходят в состояние ожидания. Это ожидание продолжается два такта (пятый и шестой) до завершения первой команды.

Обратите внимание! Вторая команда выполняется всего за три такта, и она могла бы быть завершена на пятом такте, но ей приходится ждать завершения первой, так как она началась позже. Аналогичным образом уже фактически выполненная к шестому такту третья команда ожидает завершения первых двух.

Первая команда занимает шесть тактов, поэтому на седьмом такте регистр R3 освобождается и четвертая команда поступает на конвейер, который возобновляет свою работу и без остановок обрабатывает команды в течение трех тактов. На десятом такте в седьмой команде опять обнаруживается RAW-зависимость от регистра R1 и конвейер вновь останавливается.

На рис. 11.7 показаны временны́е диаграммы выполнения команд из табл. 11.1. Сплошной линией отмечены такты выполнения действий, а пунктирной — такты ожидания. Если пунктирная линия расположена перед сплошной, команда ожидает освобождения ресурса, от которого она зависит. Если пунктирная линия расположена после сплошной, команда ожидает завершения предыдущей. На рисунке наглядно видно, что конвейер вынужден простаивать довольно много времени.

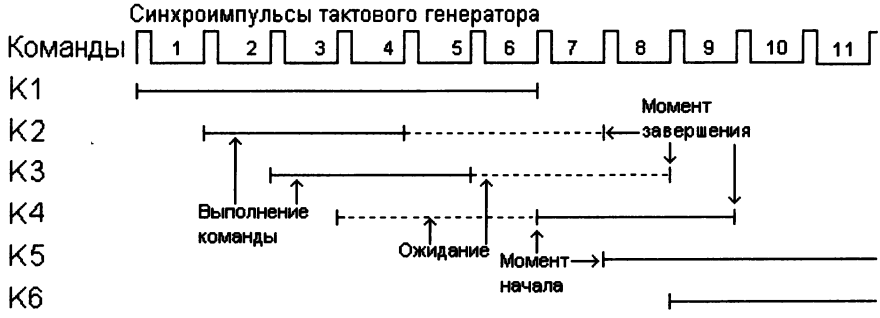


Рис. 11.7. Выполнение команд в программном порядке

Однако порядок следования машинных команд в программе может быть изменен без всякого ущерба для конечного результата. Очевидно, что менять местами можно только взаимно независимые команды. На этом наблюдении базируется метод анализа потока данных, в котором порядок выполнения машинных команд программы переопределяется так, чтобы, не нарушая логики работы программы, получить более эффективную загрузку конвейера. При обнаружении взаимозависимости предлагается задерживать передачу на конвейер не всех команд программы, а только тех, которые зависят от уже находящихся на конвейере, и начинать выполнение любых последующих независимых команд. Единственное накладываемое на переопределение порядка команд ограничение состоит в том, чтобы конечный результат не зависел от способа переупорядочения и совпадал с результатом естественного порядка выполнения команд, полученным с приостановками конвейера.

В табл. 11.2 приведен пример выполнения той же самой программы, но с переопределением порядка машинных команд, а на рис. 11.8 изображена соответствующая этой таблице временная диаграмма выполнения команд.

Порядок выполнения трех первых команд, в которых нет взаимозависимостей, не претерпел изменений по сравнению с естественным порядком.

Таблица 11.2. Выполнение машинных команд с переопределением порядка

Такт	Номер команды	Действие	Поступление	Длительность	Завершение
1	1	R3:=R0×R1	1	6	
2	2	R4:=R0+R2	2	3	
3	3	R5:=R0+R1	3	3	
4	5	R7:=R0×R8	5	6	
5	6	S1:=R0×R0	6	6	2
6	4	R6:=R1+R3	4	3	3
7					1
…	…	…	…	…	…
	7	R3:=S1–R2		3	

Рис. 11.8. Выполнение команд с переупорядочением

На четвертом такте должно начаться выполнение четвертой команды, имеющей взаимозависимость с первой. Ее поступление на конвейер откладывается, и вместо нее на конвейер подается пятая команда.

На пятом такте четвертая команда все еще не может попасть на конвейер. Следующая, шестая команда, R1:=R0×R0, должна записать новый код в регистр R1. А читающая из регистра R1 четвертая команда отложена до освобождения R3 в шестом такте. Следовательно, запись в регистр R1 нового кода до того, как выполнится четвертая команда, приведет к искажению результата. В этой ситуации имеется WAR-зависимость поступившей вне очереди команды от задержанной команды. Можно, конечно, задержать и шестую команду, как задержана четвертая. Однако существует более изящный способ решения проблемы, состоящий в использовании **подмены** регистра. Процессор компьютера кроме обычных регистров, которые называются также **видимыми**, содержит определенное количество недоступных программисту регистров, которые принято называть **скрытыми**. Они автоматически используются аппаратными средствами процессора или операционной системой для временного хранения кода в ситуациях, подобных описанной. Если записать код в обычный регистр по каким-либо причинам невозможно, аппаратура (или операционная система) автоматически производит запись в соответствующий скрытый регистр. Когда обычный регистр освобождается, автоматически производится перезапись кода из скрытого регистра в освободившийся обычный. В табл. 11.2 скрытым является регистр S1. Отметим, что подмена регистра может быть использована только при WAR- или WAW-зависимости. Для RAW-зависимости способ не может быть использован.

Наконец, на шестом такте четвертая команда попадает на конвейер. Следом за ней должна была бы пойти седьмая команда, но ее передача на конвейер откладывается из-за зависимости от четвертой. Если в момент, когда седьмая команда R3:=R1−R2 поступит на конвейер, обратная подмена скрытого регистра S1 обычным регистром R1 еще не произойдет, в команде выполнится чтение из скрытого регистра R3:=S1−R2 (см. последнюю строку табл. 11.2).

Теперь обратим внимание на последний столбец табл. 11.2. Из него видно, что первой с конвейера на пятом такте сходит вторая команда, а за ней (на шестом такте) — третья. И только после них (на седьмом такте) конвейер покидает первая команда. Видно, что порядок завершения команд не соответствует порядку

их поступления на конвейер. Метод изменения последовательности команд сопровождается сохранением в специальной таблице их нового порядка. Поэтому нет нужды задерживать выпуск с конвейера команды до завершения ранее начатой. Если возникнет прерывание, точную обстановку можно восстановить с помощью этой таблицы.

Таким образом, использование переупорядочения команд и подмены регистров обеспечивает существенное увеличение производительности конвейера. Даже по рассмотренному фрагменту видно, что на каждом такте на конвейер поступает новая команда, а выполнение первых трех команд вместо девяти занимает только семь тактов.

11.4.2. Предсказание перехода

Рассмотрим более подробно проблему конвейерной организации, связанную с наличием в программе команд перехода. Пусть в некоторый момент времени блок выборки извлек из буфера команду перехода, например команду `jl vr2` (см. рис. 4.39). В этот момент ни одному из блоков процессора еще не известно, что это команда перехода. Как и любой другой код линейного участка, на следующем такте код команды перехода передается блоку декодирования, а блок выборки должен извлечь следующую команду. Как показано в разделах 4.3.10–4.3.11, сразу после команды перехода в программе обычно находятся команды, начинающие одну из ветвей ветвления, или команды участка, расположенного за циклом. В примере на рис. 4.39 это команда `mov bx,R2`, принадлежащая ветви с *невыполненным* условием. К тому времени, когда станет известно, какая именно ветвь должна исполняться, на конвейере уже будут обрабатываться несколько команд этой ветви или этого участка. И если на нем оказывается не та ветвь, которая нужна, конвейер придется освободить и повторно заполнить командами другой ветви.

Простейший способ решения этой проблемы, который использовался в первых конвейерных процессорах, состоит в опережающем просмотре команд и в передаче на конвейер команд `nop` (нет операции) после любой обнаруженной команды перехода. Это эквивалентно простою процессора в течение нескольких тактов. Неэффективность этого подхода очевидна.

Чтобы уменьшить время простоя, предложены различные алгоритмы прогнозирования выбора ветви. В частности, анализ выполнения процессором программ показывает, что во время выбора ветви с большей вероятностью выбирается та же самая ветвь, которая выбиралась при предыдущем прохождении этого ветвления. Еще проще ситуация в переходе, связанном с циклом, так как переход назад выполняется во всех случаях, кроме одного, завершающего цикл. Обсуждаемый алгоритм предсказания ветвлений базируется на фиксации выбранной однажды ветви ветвления. При каждом новом прохождении точки ветвления прогнозируется выбор той же самой ветви. На основании этого прогноза конвейер заполняется командами из этой ветви. Если прогноз оправдывается, то оказывается, что конвейер продолжает работать в установившемся режиме, как на линейном участке. И только в случае неверного предсказания конвейер приходится очищать.

Практика показывает существенное повышение эффективности конвейера даже при таком простейшем алгоритме прогнозирования.

Реализация этого варианта прогнозирования базируется на таблице, в которой для каждой команды перехода фиксируется адрес перехода и бит, значение которого показывает, был или не был выбран этот адрес для перехода при последнем выполнении команды.

Существуют и более надежные алгоритмы предсказания переходов, основанные на других подходах. Например, более развитая технология **множественного предсказания ветвлений** основана не на фиксации последней выбранной ветви, а на подсчете частоты выбора каждой из ветвей ветвления и предсказании прохождения программы по ветви с максимальной частотой. Процессор может предвидеть разделение потока команд, в том числе с помощью просмотра программы на несколько шагов вперед. Это дает возможность с 90-процентной точностью предсказывать ветвь, которая будет выбрана.

11.4.3. Спекулятивное выполнение

Для повышения эффективности конвейера при наличии ветвлений может быть использован еще один способ, который называется спекулятивным выполнением. Его суть состоит в выполнении команды заранее, до того как станет известно, нужно ли ее вообще выполнять. Обычно это команды, расположенные за командой условного перехода. Рассмотрим, например, ветвление из раздела 4.3.10 (см. рис. 4.34):

```
jl  vr2
mov bx,R2
jmp cont
vr2 mov bx,R1
cont mov R3,bx
```

До фактического выполнения команды `jl vr2` неизвестно, какая именно команда (`mov bx, R1` или `mov bx, R2`) будет выбрана. В данном случае спекулятивное выполнение означает, что параллельно выполняются обе ветви, причем их выполнение с помощью переупорядочения может быть начато даже до команды перехода. Для записи результатов в случае спекулятивного выполнения обычно используется подмена регистров, например, `mov S1, R1` и `mov S2, R2`. После того как выяснится, какая именно ветвь нужна, производится соответствующая обратная подмена скрытого регистра `S1` или `S2` регистром `bx`. Ненужный результат при этом отбрасывается. Ясно, что такая организация вычислений возможна при определенном избытке аппаратных ресурсов, которые могут быть использованы в том числе и для бесполезной с точки зрения конечного результата работы.

Реализация спекулятивного выполнения может быть связана с обработкой прерывания в одной из спекулятивных ветвей. Немедленная обработка такого прерывания нецелесообразна, так как неизвестно, будет ли выбрана вызвавшая прерывание

ветвь. Одно из возможных решений этой проблемы — отложенная реакция на прерывание. Возникновение прерывания фиксируется, но его обработка откладывается до момента обнаружения полезности вызвавшей прерывание команды.

Реализация спекулятивного выполнения особенно удобна в процессорах, имеющих суперскалярную архитектуру. Разные ветви передаются на выполнение различным исполнительным блокам конвейера, которые выполняют их одновременно. Это обеспечивает максимальную загруженность процессора и увеличивает скорость исполнения программы.

11.4.4. Многопоточное исполнение

В суперскалярных и суперконвейерных архитектурах в состав процессора входит несколько исполнительных блоков, которые могут одновременно выполнять операции, заданные в командах выполняющейся программы. Вместе с тем, понятно, что достичь одновременной занятости всех исполнительных блоков, которая обеспечивает максимальную производительность процессора, возможно далеко не всегда. Такая загрузка существенно зависит от выполняемой программы, от качества трансляции и множества других факторов. Как показывает анализ, реальная загруженность исполнительных блоков процессора в суперскалярной архитектуре при выполнении одиночной программы в среднем достигает только 35 %.

В многопрограммном режиме процессор обслуживает несколько потоков команд. Эти команды могут принадлежать разным программам или одной и той же многопоточной программе. Напомним, что многопоточной считается программа, имеющая несколько ветвей — потоков, которые логически могут выполняться одновременно. Такие ветви организуются специальными средствами языка программирования. Во время выполнения многопоточной программы для каждого потока создается подчиненная задача (подзадача), которая с точки зрения ее выполнения процессором не отличается от независимой задачи.

Наличие нескольких потоков команд, принадлежащих одной и той же или разным программам, ситуацию с загруженностью исполнительных блоков процессора не улучшает. Дело в том, что процессор предоставляется каждой задаче или подзадаче на некоторое количество тактов, в течение которых на конвейере могут находиться команды, принадлежащие только одному потоку. На рис. 11.9, *а* изображена условная ситуация, когда процессор содержит только два исполнительных блока, Блок 1 и Блок 2. Пусть процессор предоставляется каждому потоку на 10 тактов. Тогда суммарное время выполнения изображенных на рисунке участков потоков составляет 20 тактов процессора.

Горизонтальные линии показывают такты, в которых исполнительные блоки заняты. Отсутствие линии означает, что блок простаивает. Несмотря на характерное для суперскалярной архитектуры совмещение во времени работы исполнительных блоков процессора, видно, что в течение нескольких тактов на каждом из потоков простаивает либо один блок, либо другой. Следовательно, наличие нескольких потоков, выполняющихся в режиме переключения процессора, не приводит к увеличению отдачи от процессора по сравнению с обработкой одного потока.

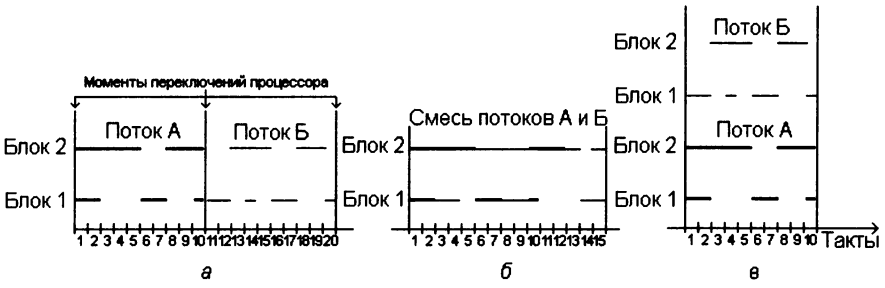


Рис. 11.9. Упрощенная схема выполнения потоков: *а* — в обычном режиме; *б* — в режиме Hyper Threading; *в* — на многоядерном процессоре

В 2002 г. компания Intel предложила технологию, получившую название **Hyper Treading** (увеличенный поток, или сверхпоток). Основная идея этого подхода состоит в том, чтобы допустить возможность одновременного выполнения исполнительными устройствами процессора команд из различных потоков. На рис. 11.9, *б* видно, что простаивавшие в режиме переключения блоки оказываются занятыми выполнением команд из другого потока (чтобы различить команды разных потоков, линии потока А изображены более толстыми по сравнению с линиями потока Б). В результате общее время выполнения одних и тех же участков потоков А и Б оказывается меньше, чем в режиме переключения между потоками. Прямые тестовые испытания показывают, что использование этой технологии приводит к увеличению средней производительности примерно на 30 %.

В некотором смысле можно считать, что работа одного физического процессора по технологии Hyper Threading эквивалентна одновременной работе двух логических процессоров, каждый из которых имеет те же характеристики производительности, что и физический процессор. Отметим, что даже при использовании технологии Hyper Threading достичь стопроцентной загрузки всех исполнительных блоков процессора невозможно.

11.5. Многопроцессорные и многоядерные архитектуры

Дальнейшее наращивание производительности однопроцессорного компьютера, вероятнее всего, будет происходить с большими трудностями и вскоре станет невозможным.

Вместе с тем разработчики уже довольно давно начали создавать вычислительные системы, которые содержат несколько процессоров, обладающих возможностью одновременно выполнять разные участки одной и той же или различных программ. Такие *многопроцессорные* системы имеют ряд очевидных преимуществ перед однопроцессорными. Во-первых, несколько процессоров могут выполнить больше работы за фиксированный промежуток времени или же выполнить ту же работу, но быстрее, чем один. Во-вторых, многопроцессорная система надежнее, чем однопроцессорная, так как вышедший из строя процессор может быть почти

мгновенно заменен другим процессором без остановки работы системы. В-третьих, такая система может одновременно обслужить большее количество пользователей с меньшим временем ожидания ответа для каждого из них. Имеются и другие выгодные моменты, поэтому создано множество различных вариантов многопроцессорных систем, в которых разработчики пытались реализовать эти преимущества. К настоящему времени построены многопроцессорные системы, состоящие из сотен тысяч процессоров и обладающие огромными вычислительными мощностями, в миллионы раз превосходящими производительность любых однопроцессорных систем.

Однако системы, содержащие несколько процессоров, стоят гораздо дороже, чем системы с одним процессором. Кроме того, у многопроцессорных систем существует множество конструктивных проблем, связанных с организацией электропитания, охлаждения, обслуживания, с геометрическими размерами и т. д. Многопроцессорным системам нужно специальное программное обеспечение, которое должно учитывать наличие нескольких процессоров. Операционные и инструментальные системы для них должны обеспечивать оптимальное распределение между процессорами потоков команд и данных, разрешать конфликты при одновременном запросе одного и того же ресурса, поддерживать когерентность данных в кэшах процессоров, а также решать множество других задач параллельной обработки данных. Более подробно особенности архитектуры многопроцессорных вычислительных систем обсуждаются в главе 17.

Современные процессоры достигли тактовых частот порядка нескольких гигагерц. На частоте 2 ГГц такт длится 0,5 нс, за это время электромагнитное поле распространяется на расстояние всего 15 см. Данные оценки показывают одно из узких мест многопроцессорных систем. Существующие ограничения на минимальные конструктивные расстояния между процессорами, вызванные рядом технических и физических факторов, например необходимостью обеспечивать охлаждение процессоров и защищать от взаимного искажения передаваемые по шинам данные, начинают сказываться на скорости обработки данных. Если, например, процессоры находятся на расстоянии 0,5 м друг от друга, то обмен данными между ними приведет к задержке на четыре такта, в течение которых биты проходят по линиям шины от одного процессора к другому.

По этой причине разработчики в последние годы занимались разработкой систем, которые содержат два процессорных ядра в одной микросхеме процессора. Каждое ядро по своим функциональным возможностям полностью аналогично обычному одноядерному процессору. Другими словами, двухъядерный процессор соответствует вычислительной системе с двумя отдельными процессорами. Различие проявляется в том, что в двухъядерном варианте эти процессоры конструктивно размещаются *предельно близко*, на площади примерно 200–300 мм^2, что обеспечивает значительную экономию на времени передачи данных по сравнению с двухпроцессорным вариантом. Одновременно снимается ряд других конструктивных проблем: например, теперь достаточно всего одной, но более мощной системы охлаждения.

Использование многоядерных процессоров обеспечивает более высокую производительность системы и по сравнению с одноядерным процессором, работающим с применением технологии Hyper Threading. На рис. 11.9, *в* изображена ситуация, при которой одно ядро занято выполнением команд потока A, а второе выполня-

ет команды потока Б. При этом общее время выполнения обоих потоков оказывается равным 10 тактам, то есть времени выполнения одного потока.

Отметим, что каждое из ядер процессора работает по технологии Hyper Threading, по этой причине работа одного двухъядерного процессора может рассматриваться как одновременная работа четырех логических процессоров.

Итак, многоядерные процессоры решают проблему ограничений на минимальные расстояния между процессорами и увеличивают общую производительность системы. Однако остальные проблемы многопроцессорных систем, которые связаны с программным обеспечением параллельных вычислений, характерны и для систем с многоядерными процессорами, и их еще предстоит решать. Предполагается, что в ближайшие годы это направление развития вычислительных систем станет одним из основных.

Дополнительный материал по обсуждавшимся в главе вопросам можно найти в изданиях [18], [30], [34], [35].

Контрольные вопросы и упражнения

1. Дайте определения технических характеристик микропроцессора.
2. Охарактеризуйте микроархитектурный уровень структуры процессора.
3. Поясните смысл терминов «микрокоманда» и «микропрограмма».
4. Какие микрокоманды можно выделить в реализации команды умножения?
5. Опишите конвейерный режим функционирования процессора.
6. Поясните смысл терминов «конвейер», «ступень конвейера», «длина конвейера».
7. Оцените возможное ускорение работы процессора от использования конвейера.
8. Какая архитектура считается суперконвейерной?
9. Почему нельзя включить в состав процессора большое количество конвейеров?
10. Опишите суперскалярную архитектуру процессора.
11. Дайте сравнительную характеристику суперконвейерной и суперскалярной архитектур.
12. Какие проблемы возникают в конвейерной организации процессоров?
13. Из каких элементов складывается динамическое исполнение машинных команд?
14. Какие взаимозависимости существуют между машинными командами программ?
15. Почему машинные команды должны покидать конвейер в том же порядке, в котором они поступили на него?
16. В чем состоит метод изменения порядка следования команд?
17. Что дает предсказание ветвления?
18. В чем состоит спекулятивное выполнение?
19. Сравните обычный и многопоточный способы выполнения машинных команд.
20. Дайте сравнительную характеристику многопроцессорных и многоядерных систем.

Глава 12
Внешняя память

Как отмечалось в 4.1.4, внешними запоминающими устройствами (ВЗУ), или просто внешней памятью (ВП), в компьютере считается группа устройств, которые предназначены для долговременного хранения больших массивов программ и данных. Отличительными особенностями внешней памяти являются:

- отсутствие доступа со стороны процессора;
- энергонезависимость;
- малая скорость обмена (относительно оперативной памяти);
- практически неограниченный объем;
- относительная дешевизна хранения данных.

В настоящее время основными разновидностями внешней памяти являются гибкие и жесткие магнитные диски, оптические диски, магнитные ленты и переносимые устройства памяти.

12.1. Магнитные диски

Магнитный диск состоит из нескольких насаженных на общую ось круглых пластин, сделанных из винила, алюминия, стекла и некоторых других материалов. Диаметр первых магнитных дисков составлял 50 см. В настоящее время диски имеют диаметр от 3 до 12 см в зависимости от модели. Диск для портативных компьютеров, а также для цифровой видео- и аудиотехники имеет диаметр менее 3 см.

На плоские поверхности диска наносится специальное вещество, которое с высокой степенью надежности сохраняет свое состояние намагниченности. Такая поверхность диска называется **рабочей**. Очевидно, что одна пластина диска может иметь одну или две рабочие поверхности. Каждая рабочая поверхность имеет номер, причем нумерацию поверхностей принято начинать с нуля. На рис. 12.1

изображена схема диска, состоящего из двух пластин и имеющего четыре рабочие поверхности.

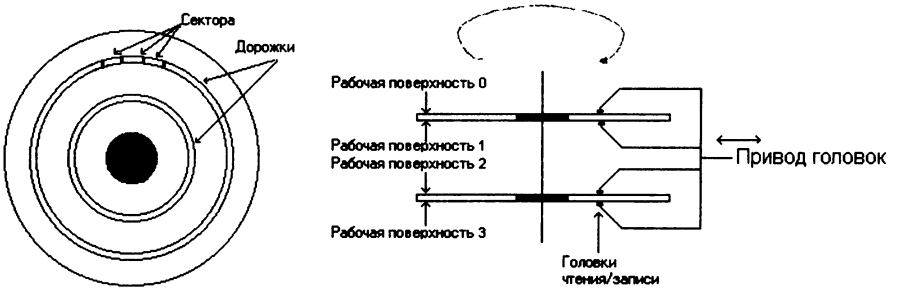


Рис. 12.1. Структура магнитного диска

Над каждой рабочей поверхностью вращающегося диска находится закрепленная на **приводе** головка чтения/записи, содержащая индукционную катушку. Привод представляет собой механизм, обеспечивающий одновременное радиальное перемещение всех головок чтения/записи и точное их позиционирование на некотором расстоянии от центра диска. Сочетание вращательного движения пластин и радиального перемещения головок обеспечивает возможность чтения/записи в любой точке рабочих поверхностей.

В режиме записи при прохождении по индукционной катушке тока определенного направления на рабочей поверхности остается участок с соответствующим направлению тока направлением намагниченности. Одно из этих направлений считается кодом нуля, другое — кодом единицы. В режиме чтения, когда головка проходит над намагниченной областью, в ней возникает ток соответствующего намагниченности направления, который воспринимается как прочитанное значение бита. Линейный размер считающейся битом области намагниченности составляет примерно 0,1–0,2 мкм.

Дорожкой диска называется кольцевая область рабочей поверхности диска, содержащая последовательность битов, записанных на его рабочую поверхность за один полный оборот диска (см. рис. 12.1). Ширина дорожки составляет 5–10 мкм в зависимости от модели. Каждая дорожка имеет собственный номер; нумерация дорожек, так же как и нумерация рабочих поверхностей, начинается с нуля. Группа дорожек, имеющих один и тот же номер, но расположенных на разных рабочих поверхностях, образует **цилиндр**.

Из-за того, что длины дорожек, находящихся на различном удалении от центра диска, различны, а количество битов на всех дорожках одинаково, плотность записи битов на различных дорожках различна. Чем ближе к центру дорожка, тем более высокая плотность записи используется.

Дорожка состоит из **секторов** обычно фиксированной длины (рис. 12.2). Между двумя соседними секторами дорожки всегда имеется пустой **межсекторный** интервал. Его наличие вызвано причинами технического характера, в частности, конечным временем отключения и включения полезного сигнала.

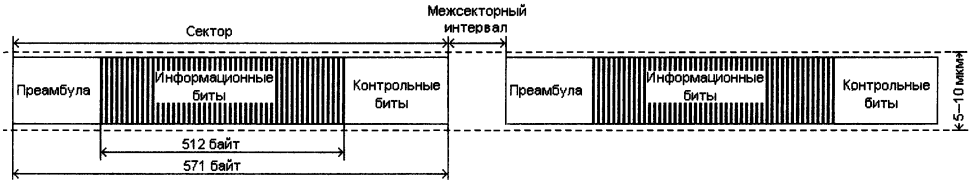


Рис. 12.2. Структура стандартной дорожки магнитного диска

Каждый сектор имеет номер, но он начинается с единицы. Стандартный сектор дорожки содержит 512 байт данных. Перед записанными в секторе данными находится **преамбула**, представляющая собой специальный код, позволяющий головке выполнить синхронизацию перед началом чтения или записи. В частности, в преамбуле находится порядковый номер, по которому опознается нужный сектор. Из-за малых линейных размеров бита необходимо очень точно позиционировать головку чтения/записи над дорожкой, при этом высокая плотность записи кодов на рабочую поверхность и высокая скоростью вращения обусловливают высокую вероятность появления ошибки чтения или записи. Поэтому в каждый сектор включаются контрольные биты кода Хемминга или более развитого кода Рида — Соломона, которые обеспечивают исправление нескольких одновременно появившихся в секторе ошибок. Общая длина сектора вместе с преамбулой, информационными и контрольными битами составляет 571 байт. Многие производители дисков указывают в маркировке или характеристике диска его полную емкость, как будто весь сектор может быть занят информационными битами и нет ни битов преамбулы, ни контрольных битов кода Хемминга. Из-за этого емкость диска на его маркировке и емкость, показываемая операционной системой, не совпадают — последняя меньше примерно на 15 %.

12.1.1. Гибкие магнитные диски

Гибкий магнитный диск **(ГМД)**, накопитель на гибких магнитных дисках **(НГМД)**, **флоппи**-диск (от floppy disk — свободно висящий диск), или просто **дискета**, представляет собой находящуюся в твердом защитном чехле гибкую лавсановую пластинку диаметром 3,5 дюйма, что примерно равно 9 см (1 дюйм равен 2,54 см). Заметим, что ранее использовались гибкие диски размером 8 и 5 дюймов.

Рабочие характеристики гибких дисков: емкость 1,44 Мбайт, количество дорожек на рабочей поверхности и, следовательно, количество цилиндров — 80, количество секторов на дорожке — 18, количество рабочих головок — 2. Диск вращается со скоростью 300 об./мин, обеспечивая скорость передачи данных 500 Кбит/с.

Для работы с дискетами в компьютере предусмотрены устройства, которые называются **дисководами** гибких магнитных дисков, или **FDD** (от floppy disk drive — привод флоппи-дисков). Во время вращения диска в дисководе головка чтения/записи касается рабочих поверхностей, что приводит к быстрому износу дискеты. Чтобы уменьшить износ, в нерабочее время дисковод останавливает вращение и отводит головки в нейтральную позицию. Такое периодическое включение и выключение вращения сказывается на надежности дисковода.

Следует также отметить, что удельная стоимость хранения одного байта на гибких дисках по сравнению с другими видами магнитных носителей (а тем более оптических) очень высока. Перечисленные факторы приводят к тому, что дискеты постепенно выходят из употребления. Тем не менее некоторые производители пытаются сохранить гибкие диски в качестве сменных носителей информации, улучшая их характеристики. Например, в конце 2000 г. фирма Panasonic разработала дисководы стандарта FD32MB, которые обеспечивают запись на стандартную трехдюймовую дискету 32 Мбайт данных.

12.1.2. Жесткие магнитные диски

Жесткий магнитный диск (ЖМД) называют также **винчестером** (от Winchester — разновидность двуствольной винтовки) и **HDD** (от Hard Disk Drive — привод жесткого диска). Он представляет собой пакет, содержащий от 2 до 10 закрепленных на общей оси пластин. Пакет помещается в герметичный корпус, в котором создается разрежение воздуха, обеспечивающее высокую скорость вращения и передачи данных в процессе обмена. Во время вращения головки чтения/записи не касаются рабочих поверхностей дисков. Такая конструкция позволяет увеличить плотность записи кодов данных до 10^5 бит/см^2 и, следовательно, увеличить общий объем диска. Обычно жесткий диск является постоянным, несъемным устройством компьютера, поэтому доступ к нему без разбора корпуса компьютера обычно невозможен. Но существуют и варианты комплектации компьютера со сменными жесткими дисками.

Жесткие диски описываются следующим набором характеристик:

- объем диска;
- среднее время доступа;
- скорость вращения диска;
- скорость передачи данных;
- стандарт интерфейса;
- размер собственной кэш-памяти диска;
- форм-фактор;
- фирма-производитель.

Современные винчестеры имеют объем от нескольких десятков до нескольких сотен гигабайт. По-видимому, в ближайшее время этот показатель может возрасти до нескольких терабайт.

ПРИМЕЧАНИЕ

Относительно недавно были созданы устройства, которые способны определять в сотни раз более слабое магнитное поле, чем это было возможно до сих пор. Одним из результатов применения подобных разработок может стать повышение емкости жестких дисков в компьютерах до нескольких сотен петабайт (то есть до сотен тысяч терабайт). Заметим, что, по некоторым подсчетам, объем всей имеющейся в мире на сегодняшний день информации (включая литературу, звукозаписи и видеозаписи) не превышает 100 Пбайт.

Для перемещения головок чтения/записи к различным дорожкам требуется различное время, так как они имеют различные радиусы. Кроме того, в процессе обмена головки перемещаются между дорожками, вообще говоря, случайным образом. Поэтому для характеристики временны́х затрат, связанных с позиционированием головок над нужной дорожкой, используется величина, равная среднему времени доступа к дорожке, вычисленному по всем дорожкам рабочей поверхности. Жесткие диски имеют среднее время доступа, равное 7–9 мкс.

Пакет пластин вращается в герметичном корпусе с большой скоростью. В настоящее время стандартные скорости вращения жесткого диска составляют 5400, 7200 и 10 800 об./мин. Высокая скорость вращения дисков приводит к их нагреванию. Вследствие расширения материала диска при нагревании увеличиваются его геометрические размеры. Так как линейный размер бита на рабочей поверхности не превышает 0,2 мкм, а ширина дорожки не превышает 10 мкм, даже малые изменения размеров пластины могут привести к неточности позиционирования головок диска над битом при чтении или записи. В связи с этим некоторые диски нуждаются в рекалибровке, то есть повторной калибровке механизмов перемещения головок, учитывающей изменение геометрических размеров пластин. На выполнение рекалибровки требуется некоторое время, в течение которого диск не может быть использован для операций обмена. Это может привести к трудностям при использовании мультимедийных программ, требующих непрерывного потока битов для качественного воспроизведения звука и видео. Поэтому для хранения мультимедийных данных используются специальные аудио- и видеодиски, не испытывающие термических расширений и тем самым обеспечивающие высокое качество записи и воспроизведения звука и изображения.

Скорость чтения/записи данных у жестких дисков зависит от скорости вращения и плотности записи. Для ее указания используется спецификация вида **UDMA/n**, которая определяет максимальную скорость передачи данных в режиме DMA (прямого доступа диска к оперативной памяти), равную *n* мегабайт в секунду. Например, спецификация UDMA/100 означает, что максимальная скорость передачи равна 100 Мбайт/с.

Так как данные из файла обычно разбросаны по разным секторам, дорожкам и поверхностям диска, существует большая разница между максимально возможной скоростью передачи и фактической или средней скоростью. В настоящее время средняя скорость передачи равна 10–15 Мбайт/с.

Работой каждого диска управляют соответствующие его модели контроллеры. Каждый контроллер по определенным правилам взаимодействует с управляемым диском и с шиной, к которой он подключен. Совокупность таких правил образует стандарт, или **интерфейс**, диска и его шины. В настоящее время стандартно используемыми являются интерфейсы IDE, EIDE, SCSI и RAID.

Диски стандарта **IDE** (от Integrated Drive Electronics — интегрированное электронное устройство) имеют встроенный в корпус устройства контроллер. Таким образом, приобретая устройство, потребитель одновременно приобретает и соответствующий контроллер. В базовом стандарте диск IDE может содержать до

16 рабочих поверхностей, до 1024 цилиндров и до 63 секторов на дорожке. Общий объем таких дисков не превышает 512 Мбайт. Интерфейс IDE допускает малую и среднюю скорости обмена от 2,1 до 8,3 Мбайт/с. По этому стандарту к контроллеру можно подключить только один монопольно работающий с шиной диск.

Интерфейс **EIDE** (от Extended IDE — усовершенствованный IDE) предусматривает возможность одновременного подключения к контроллеру до четырех дисков, которые используют подключение монопольно. Это значит, что шина и контроллер этого стандарта работают аналогично селекторному каналу. К контроллеру может быть подключено несколько устройств, но любое из них захватывает шину и контроллер на весь период обмена. Контроллер переходит к обслуживанию следующего диска только после полного завершения обмена для текущего диска. Возможная скорость обмена для дисков EIDE увеличена до 20 Мбайт/с, а объем может достигать сотен гигабайт.

Интерфейс **SCSI** (от Small Computer System Interface — интерфейс малых вычислительных систем) предусматривает скорость обмена до 80 Мбайт/с при высокой стабильности передачи данных. Для дисков этого стандарта необходим отдельно приобретаемый и устанавливаемый контроллер, который допускает одновременное подсоединение к нему до семи дисковых устройств. Отличительной особенностью контроллера SCSI является возможность *одновременной* работы всех подключенных дисков. В этом смысле работа контроллера SCSI похожа на работу мультиплексного канала: несколько устройств могут начать и одновременно выполнять обмен.

Имеется несколько модификаций обсуждаемого стандарта. Например, простой стандарт SCSI предусматривает работу с шиной разрядностью 8 бит, тактовой частотой 5 МГц и скоростью обмена до 5 Мбайт/с, а в самом развитом стандарте с названием Wide Ultra2 SCSI предусматривается шина с разрядностью 16 бит, тактовой частотой 40 МГц и скоростью обмена 80 Мбайт/с.

Перспективным направлением развития интерфейса SCSI считается интерфейс **SAS** (от Serial Attached SCSI — последовательное соединение для интерфейса SCSI), обеспечивающий скорость обмена до 300 Мбайт/с.

Интерфейс **RAID** (от Redundant Array of Inexpensive Disks — избыточный массив недорогих дисков, позднее возникла трактовка Redundant Array of Independent Disks — избыточный массив независимых дисков), по сути, относится не к дискам, а к контроллеру, управляющему группой дисков. Компьютер комплектуется группой обычно SCSI-дисков, которые подсоединяются к RAID-контроллеру. Он обеспечивает восприятие операционной системой группы дисков как одного диска очень большой емкости — как диска типа **SLED** (от Single Large Expensive Disk — одиночный большой дорогой диск). Один контроллер RAID может управлять группой, содержащей до 16 дисков SCSI. Основное преимущество интерфейса RAID — возможность осуществления параллельного обмена со всеми дисками массива.

Существует несколько способов размещения данных на RAID дисках, которые обеспечивают, с одной стороны, высокую скорость передачи данных, а с другой —

высокую надежность их хранения. Эти способы принято называть **типами**, или **уровнями**, RAID. По сути дела, многие из этих способов воспроизводят на уровне жестких дисков приемы, использованные для повышения эффективности оперативной памяти.

Рассмотрим особенности некоторых вариантов организации RAID-массивов. Нулевой тип (нулевой уровень) массива RAID аналогичен обсуждавшемуся ранее расслоению оперативной памяти (см. 7.4.1). На жестких дисках выделяются зоны, которые можно рассматривать как аналоги блоков оперативной памяти в механизме расслоения. Каждая зона включает некоторое количество секторов диска. Количество зон на диске и секторов в одной зоне может изменяться в широких пределах и подбирается программным обеспечением, исходя из скорости передачи, объема дисков, количества дисков в массиве и т. д. На рис. 12.3 изображен массив из четырех дисков, на которых организовано 16 зон.

Диск 0	Диск 1	Диск 2	Диск 3
Зона 0	Зона 1	Зона 2	Зона 3
Зона 4	Зона 5	Зона 6	Зона 7
Зона 8	Зона 9	Зона 10	Зона 11
Зона 12	Зона 13	Зона 14	Зона 15

Рис. 12.3. Массив дисков RAID нулевого типа

Каждая следующая зона располагается на следующем диске массива. Вся совокупность зон образует кольцо, то есть после записи последней зоны происходит возврат к первой. Такое распределение зон по нескольким дискам называется **разметкой**. Записываемый на диски поток байтов (или слов) направляется в разные зоны: первый байт (слово) записывается в первую зону, второй байт (слово) — во вторую и т. д. Это позволяет организовать параллельный высокоскоростной обмен с дисками группы. Но надежность всей системы дисков ниже, чем при использовании одного диска. В самом деле, вероятность выхода из строя одного из, скажем, четырех дисков в четыре раза выше, чем вероятность выхода из строя единичного диска. А при выбранной схеме разметки выход из строя любого из дисков означает, что все данные будут потеряны.

В массиве RAID первого типа все диски массива делятся на основные и дублирующие. Каждый основной диск массива имеет один дублирующий (рис. 12.4). Запись информации как в группу основных, так и в группу дублирующих производится по тому же принципу, что и для массива дисков нулевого уровня, — по разнесенным между дисками зонам. Но каждая зона записывается дважды: один раз на основной диск, а второй — на дублирующий. При этом запись зоны на дублирующий диск производится одновременно с ее записью на основной диск. Поэтому скорость передачи данных при записи такая же, как в обычном режиме. Но чтение может производиться с любого экземпляра диска, как основного, так и дублирующего. Так как обмен со всеми дисками может производиться одновременно, может быть получен выигрыш в скорости чтения в два раза.

Добавим, что эта схема имеет высокую отказоустойчивость, поскольку возможно полное восстановление информации с уцелевшего экземпляра диска.

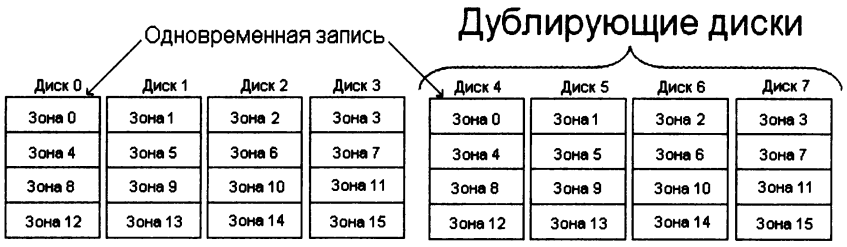


Рис. 12.4. Массив дисков RAID первого типа

В массиве RAID второго типа запись на диски производится не зонами, а полубайтами, байтами или словами, к которым добавляется необходимое количество контрольных разрядов кода Хемминга (см. 2.6), обеспечивающих необходимый уровень надежности записи данных. На рис. 12.5 показано, как к четырем битам полубайта добавляется три контрольных разряда кода Хемминга и каждый бит записывается на отдельный диск массива. Второй тип массива RAID, так же как и первый, обеспечивает высокую надежность, поскольку выход из строя любого диска не нарушает целостности информации. Но, в отличие от массива первого типа, накладные расходы гораздо меньше, так как полное дублирование требует большего количества дисков, чем включение в код контрольных битов. В то же время метод требует высокой синхронизации дисков и высокопроизводительного контроллера.

Рис. 12.5. Массив дисков RAID второго типа

В практике построения дисковых массивов RAID применяется несколько десятков увеличивающих скорость обмена и/или надежность хранения способов, являющихся модификациями и улучшениями описанных ранее.

Современный жесткий диск может комплектоваться собственным кэшем, объем которого доходит до 2 Мбайт. Роль кэша у жесткого диска полностью аналогична роли кэша оперативной памяти.

Форм-фактор диска описывает его геометрические размеры и форму. В настоящее время используются преимущественно три форм-фактора: стандартные (для настольных систем), для переносных компьютеров и для дисков типа Micro Drive.

Наиболее известными фирмами-производителями жестких дисков в настоящее время являются IBM, Fujitsu, Western Digital (WD), Seagate, Quantum.

12.2. Оптические диски

Оптические диски, так же как и гибкие диски, относятся к группе сменных носителей информации. Первые диски имели диаметр 30 см и использовались для хранения копий кинофильмов. В 1980 г. компаниями Philips и Sony были разработаны оптические диски меньшего диаметра. Они получили название **CD** (от Compact Disc — компактные, то есть небольшие диски). Немного позже был утвержден сохраняющийся до сих пор стандарт IS10149 технических характеристик компакт-дисков. В частности, они должны иметь диаметр 117 мм, толщину 1,2 мм и диаметр центрального отверстия 15 мм. Предполагается, что компакт-диски смогут сохранять записанную на них информацию в течение 100 лет.

12.2.1. Компакт-диски CD-ROM

Первые компакт-диски изготавливались по следующей технологии. Вначале с помощью мощного лазера в контрольном стеклянном диске выжигаются углубления диаметром 0,8 мкм, которые рассматриваются как коды единичного значения бита. Отсутствие углубления считается кодом нулевого значения бита. Углубления принято называть **впадинами**, а ровные поверхности между ними — **площадками**. Запись кода начинается от центра и продвигается к краю, при этом впадины и площадки наносятся на поверхность диска не концентрическими полосами, как у магнитных дисков, а в виде спирали (рис. 12.6), которая проходит около 22 000 оборотов вокруг диска. Общая длина спирали составляет более 5 км.

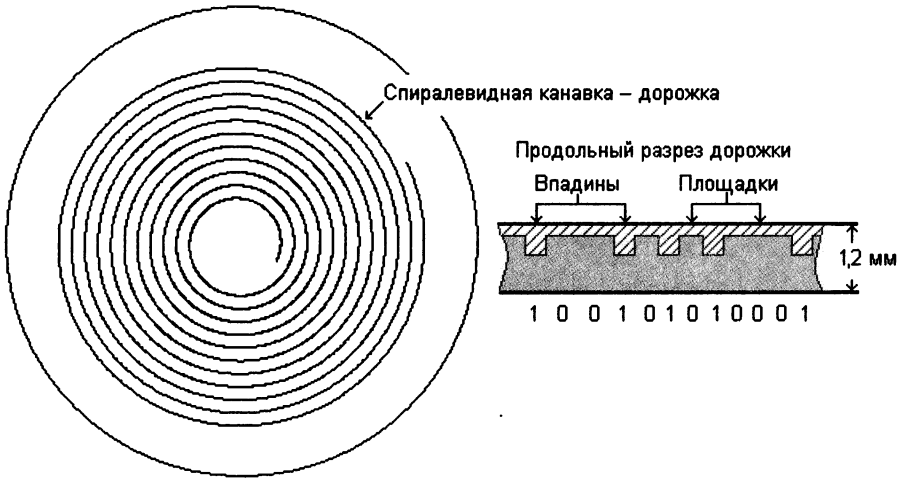


Рис. 12.6. Структура компакт-диска CD-ROM

По этому диску делается шаблон, у которого вместо углублений образуются выступы. В процессе производства компакт-диска в шаблон вводится жидкая смола (поликарбонат), в которой углубления образуются там же, где и у контрольного

диска. Далее на смолу наносится образующий жесткую основу диска тонкий слой алюминия, который покрывается защитным лаком. Очевидно, что таким способом можно получить диск, с которого можно только считывать однажды записанные на него коды. Изготовленные в условиях промышленного производства оптические диски, которые могут использоваться только для чтения, получили название CD-ROM (от Compact Disk Read Only Memory — память только для чтения на компакт-дисках).

Во время воспроизведения луч лазера небольшой мощности диаметром 0,78 мкм пробегает по спиральной канавке диска. Лазер светит с той стороны, где находится слой смолы, поэтому впадины для луча — это выступы на ровной поверхности. Вообще говоря, при считывании воспринимаются не сами выступы, а переходы между ними и ровными участками. Это обеспечивает более высокую надежность считывания. Чтобы считывание выполнялось равномерно и надежно, луч лазера должен двигаться по спирали с постоянной линейной скоростью 1,2 м/с. Поэтому угловая скорость вращения по мере считывания уменьшается от 530 до 200 об./мин. Отметим, что магнитные диски вращаются с постоянной угловой скоростью.

В связи с высокой плотностью записи информации на компакт-диски их разработчики уделяли много внимания надежности выполнения операций записи и чтения. В 1984 г. был принят международный стандарт, содержащий описание методов борьбы с ошибками хранения информации на CD-ROM. Предусмотренная в стандартах система кодирования данных включает трехуровневую защиту от ошибок.

На самом нижнем уровне каждые восемь информационных битов байта дополняются шестью контрольными битами кода Хемминга, которые позволяют исправлять двойные ошибки в одном байте. Таким образом, один байт компакт-диска фактически состоит из 14 битов.

На втором уровне защиты к каждой группе из 24 байтов данных (то есть 14-битных кодов) добавляются 18 контрольных кодов той же самой длины. Эту последовательность из 42 кодов принято называть **фреймом** (от frame — каркас). Следовательно, фрейм, состоящий из $42 \cdot 14 = 588$ битов, содержит всего $24 \cdot 8 = 192$ информационных бита.

На верхнем уровне защиты от ошибок каждые 98 фреймов образуют сектор. Сектор начинается с занимающей 16 байтов преамбулы. Ее первые 12 байтов содержат специальный код, который обеспечивает синхронизацию во время обмена. Следующие три байта преамбулы содержат порядковый номер сектора, а в ее последнем байте находится код типа диска. За преамбулой располагаются 2048 информационных и 288 контрольных байтов кода Рида — Соломона. Под преамбулу и контрольный код занята некоторая часть информационных байтов фреймов. Сектор, состоящий из 98 фреймов, имеет общую длину 57 624 бита, из них к информационным относятся только 16 384 бита. То есть информационную нагрузку несут примерно 28 % площади канавки компакт-диска, а остальные обеспечивают правильность записи и чтения.

Стандартный диск вмещает 332 800 секторов и имеет объем 650 Мбайт. Отметим, что имеются оптические диски, у которых проверки верхнего уровня

отсутствуют и информационными являются не 2048, а 2336 байтов сектора, то есть все байты сектора, за исключением преамбулы. Существуют и другие варианты стандартов, в соответствии с которыми объем компакт-диска может доходить до 800 Мбайт.

Скорость 150 Кбайт/с, характерная для первых компакт-дисков, в настоящее время принята в качестве стандартной единицы измерения скорости. Фактическая скорость обмена указывается множителем, определяющим, во сколько раз допустимая скорость чтения больше, чем базовая скорость 150 Кбайт/с. Эта характеристика указывается в названии дисков или как их основная характеристика. Если, например, в маркировке диска указано 8x–12x, это означает, что он может использоваться для работы на дисководах со скоростями от 1200 Кбайт/с до 1800 Кбайт/с. Заметим, что в настоящее время уже имеются 52-скоростные дисководы (52x), что составляет примерно 7,6 Мбайт/с.

12.2.2. Компакт-диски однократной записи CD-R

Существенный недостаток CD-ROM — невозможность записи на диск при их использовании — был устранен в дисках **WORM** (от Write Once/Read Many — однократная запись, множественное считывание), которые имеют более распространенное название **CD-R** (от Compact Disk Recordable — записываемый компакт-диск). На диски этого типа можно записывать информацию, так же как и на обычную гибкую дискету, — прямо на компьютере, *но только один раз*. Чтение такого диска может производиться произвольное количество раз. Для использования этой технологии требуются специальные заготовки дисков и специальные дисководы.

Запись двоичного кода на заготовки (в просторечии **болванки**) таких дисков производится с помощью воздействия высокотемпературного лазерного луча на особый светочувствительный слой диска из цианина зеленого цвета или пталоцианина желто-оранжевого цвета, который при этом как бы «выгорает». Упомянутые вещества сходны с красителями, используемыми в некоторых видах цветных фотопленок.

В исходном состоянии заготовки светочувствительный слой прозрачен. Запись кода выполняется с помощью луча лазера высокой мощности. Для записи кода единицы луч лазера, пробегающий по спирали над рабочей поверхностью, соприкасается со светочувствительным слоем, нагревает его и образует в месте контакта темное пятно. При записи кода нуля контакт между лучом и слоем отсутствует, и слой остается прозрачным.

Чтение кода осуществляется с помощью луча лазера, мощность которого уменьшена примерно в 20–30 раз. Пробегающий над поверхностью с закодированным темными и светлыми областями двоичным кодом луч отражается от нее и улавливается входящим в состав дисковода фотодетектором, который выявляет разницу между темными и прозрачными областями канавки, — тем самым записанный ранее код считывается.

В отличие от магнитных дисков, запись на которые производится отдельными кластерами, не обязательно размещенными последовательно друг за другом, запись файлов на оптические диски может производиться только сплошными участками. Группа последовательных секторов, записываемых за один раз, называется **дорожкой**. Еще раз подчеркнем, что дорожка компакт-диска — это не окружность, а состоящий из произвольного количества секторов участок спирали.

Для обеспечения надежной записи каждая дорожка должна записываться непрерывно без изменения скорости луча. Поэтому жесткий диск должен передавать записываемые на оптический диск данные без задержек на поиск нужного файла. Чтобы не произошла остановка во время записи дорожки на компакт-диск, программы, записывающие информацию на CD-R, вначале формируют на жестком диске так называемый **файл-образ** всех объектов, которые должны быть записаны на компакт-диск. Все предназначенные к записи на компакт-диск файлы и папки собираются в буферной области на жестком диске, и из них формируется, создается временный единый файл, который и носит название файл-образ. После чего за один сеанс записи, который принято называть **сессией**, лазерный луч переносит его на оптический диск.

Во время сессии на оптическом диске формируется *оглавление*, содержащее информацию о записанных на диск файлах и папках и их расположении. Это оглавление называется **VTOC** (от Volume Table Of Content — таблица содержания). Наличие на оптическом диске единственной VTOC позволяет организовывать единственную сессию записи. Если за время такой сессии используется не весь возможный объем диска, неиспользованный участок рабочей поверхности диска безвозвратно теряется.

Немного позднее появился более эффективный стандарт CD-R XA, в котором определены средства организации *многосессионной* записи, обеспечивающей несколько последовательных сеансов записи на диск CD-R. Этот стандарт предусматривает создание во время каждой очередной сессии нового экземпляра VTOC, который может включать в себя нужные сведения из предыдущего экземпляра. Если сведения о ранее записанных на компакт-диск файлах не включаются в новый VTOC, эти файлы оказываются как бы удаленными. Но место, которое такие файлы занимают на поверхности диска, вновь использовать невозможно. Во время чтения всегда используется самый последний экземпляр VTOC. Отметим, что некоторые программы записи на CD-R не допускают многосессионную работу.

12.2.3. Компакт-диски многократной записи CD-RW

По своим размерам, объему и внешнему виду диски **CD-RW** (от Compact Disk ReWriteable — перезаписываемые компакт-диски) ничем не отличаются от дисков CD-R и CD-ROM. Но принцип записи информации на диски CD-RW совершенно

другой: на металлическую или иную подходящую наносится рабочий слой из сплава серебра, индия, сурьмы и теллура, который под влиянием лазерного луча изменяет свое состояние, переходя из кристаллического состояния в аморфное или наоборот. Причем переход из одного состояния в другое может быть выполнен многократно. Поскольку кристаллические и аморфные состояния имеют различную отражающую способность, такие переходы создают возможность многократно записывать и читать двоичные коды данных.

Лазер дисковода CD-RW имеет три уровня мощности. При самой высокой мощности луч лазера расплавляет вещество рабочего слоя и переводит его в аморфное состояние. Так получается аналог впадины на CD-ROM. При средней мощности вещество переводится в кристаллическое состояние. Так получается аналог площадки. При самой низкой мощности лазера происходит считывание.

Как уже отмечалось, первые оптические дисководы, вошедшие в состав персональных компьютеров, выполняли обмен со скоростью 150 Кбайт/с. Исторически сложилось так, что эта скорость была выбрана в качестве базовой, основной единицы измерения скорости обмена не только для CD-ROM, но и для записываемых CD-R- и перезаписываемых CD-RW-дисков. Характеристика CD-R и CD-RW включает в себя два или три показателя: скорость обмена в режиме чтения (максимальный коэффициент), скорость для режима многократной записи (минимальный коэффициент) и скорость в режиме однократной записи. Например, маркировка 12×8×32x означает, что чтение выполняется с 32-кратной базовой скоростью, запись на CD-RW — с 8-кратной скоростью, а запись на CD-R — с 12-кратной.

12.2.4. Диски DVD

В настоящее время характерный для компакт-дисков объем 600–800 Мбайт уже считается довольно маленьким. Поэтому были разработаны другие способы записи информации, позволяющие при том же самом диаметре диска разместить на нем гораздо больше данных и программ. В частности, можно упомянуть предложенный в 2000 г. стандарт **DDCD** (от Double Density CD — компакт-диск двойной плотности), который предусматривает возможность записи на стандартный компакт-диск 1,3 Гбайт. Но и этот объем не может считаться значительным.

Наиболее соответствующими современным потребностям являются диски **DVD** (от Digital Versatile Disk — цифровой многосторонний универсальный диск). Они содержат несколько рабочих слоев, которые размещаются на одной и той же рабочей поверхности. Способ записи кода на каждый слой диска DVD такой же, как на рабочую поверхность у диска CD-ROM. Для кодирования данных используется сжимающий мультимедийный формат MPEG-2. В настоящее время используются четыре основных формата DVD-дисков:

- односторонние однослойные объемом 4,7 Гбайт;
- односторонние двухслойные объемом 8,5 Гбайт;

- двусторонние однослойные объемом 9,4 Гбайт;
- двусторонние двухслойные объемом до 17 Гбайт.

ПРИМЕЧАНИЕ

Относительно недавно фирма Constellation 3D представила свою новую разработку — диски FMD-ROM (от Fluorescent Multilayer Disk — многослойный флуоресцентный диск однократной записи), в которых предложено на одной рабочей поверхности стандартного диска размещать до ста рабочих слоев общей емкостью 150 Гбайт.

Основными отличиями дисков DVD от дисков CD являются следующие:

- впадины имеют размер 0,4, а не 0,8 мкм;
- плотность спирали 0,74 мкм между дорожками, а не 1,74 мкм;
- используется луч лазера с длиной волны 0,65 мкм вместо луча с длиной волны 0,78 мкм.

Для DVD-дисков базовой единицей измерения скорости обмена считается скорость 1,38 Мбайт/с. Таким образом, обозначение 8x в маркировке DVD диска или характеристике дисковода означает, что скорость обмена равна 11,04 Мбайт/с.

В настоящее время преобладают диски DVD-ROM, хотя уже имеются довольно надежные однократно (DVD-R) и многократно записываемые (DVD-RW) диски и устройства для их записи.

12.3. Магнитные ленты

По сравнению с магнитными и оптическими дисками магнитные *ленты*, которые также относятся к группе сменных носителей, являются очень дешевым средством хранения информации. Принцип записи информации на магнитную ленту в компьютерах ничем не отличается от используемого в бытовых кассетных магнитофонах. Существенным недостатком использования магнитных лент для хранения информации является большое время обмена с лентой, которое обусловлено в основном необходимостью перемотки ленты для достижения участка, содержащего нужные данные или программы. Поэтому чаще всего запись на магнитную ленту используется с целью архивного дублирования важной информации с жесткого диска на случай его поломки. Аналогичное дублирование на гибкие и даже оптические диски обходится значительно дороже. Следует отметить, что в настоящее время разработаны устройства на магнитных лентах объемом в десятки терабайт, что может увеличить интерес к их использованию в качестве гигантского и исключительно дешевого хранилища архивной информации.

Отметим, что устройство для записи информации на магнитную ленту в персональных компьютерах называется **стример**. Это устройство не входит в стандартный комплект компьютеров.

12.4. Мобильные носители памяти

Уже довольно давно емкость стандартных дискет (1,44 Мбайт) считается недостаточной для выполнения основных функций — временного хранения информации и ее переноса между компьютерами. Поэтому разработчики предлагают различные варианты дешевых и надежных мобильных носителей информации.

12.4.1. Мобильные дисководы

Существует несколько вариантов внешних, *мобильных дисководов* большой емкости. Выражение «мобильный дисковод» означает, что переносить между компьютерами нужно не только дискету, но и сам дисковод. В частности, можно упомянуть довольно популярные модели мобильных дисководов Iomega ZIP, Iomega JAZ и ORB Drive. Эти устройства используют сменные диски собственных, специализированных форматов, которые часто называют **ZIP-дисками**. Они имеют объем от 100 Мбайт до 2 Гбайт и допускают скорость обмена до 20 Мбайт/с.

ПРИМЕЧАНИЕ

Относительно недавно выпущены мобильные дисководы модели 2,5" Combo Portable HDD, которые используют диски диаметром 2,5 дюйма емкостью до 160 Гбайт и скоростью обмена 60 Мбайт/с.

Стоимость специализированных дисков для мобильных дисководов довольно высока. Но если учесть их высокую емкость, то использование внешних дисководов может оказаться выгоднее, чем использование оптических дисков и, тем более, стандартных трехдюймовых дискет.

12.4.2. Мобильные устройства флэш-памяти

В отличие от мобильных дисководов, мобильные устройства, основанные на флэш-памяти, не используют механических перемещений при выполнении операций чтения и записи. Кроме того, они не имеют сменных носителей. Фактически эти устройства представляют собой *микросхему памяти*, снабженную защитным корпусом и разъемом USB. Однако работа с ними после подключения к порту выполняется так же, как и с жесткими дисками. В качестве примера можно упомянуть мобильное запоминающее устройство **Jet Flash** (от jet — реактивный) объемом до 4 Гбайт и скоростью обмена до 8 Мбайт/с. Это устройство выдерживает до 1 000 000 циклов чтения/записи и обеспечивает длительность хранения данных до 10 лет.

Дополнительный материал по обсуждавшимся в главе вопросам можно найти в изданиях [30], [34], [35].

Контрольные вопросы и упражнения

1. Перечислите отличительные особенности внешней памяти компьютера.
2. Опишите общее устройство магнитного диска и структуру его дорожки.
3. Какими техническими характеристиками описывается жесткий диск?
4. Охарактеризуйте существующие стандарты интерфейсов жестких дисков.
5. Чем отличаются диски стандарта EIDE от дисков стандарта SCSI?
6. Опишите особенности дисков RAID нулевого типа. Что дает их использование?
7. Опишите особенности дисков RAID первого типа. Что дает их использование?
8. Опишите особенности дисков RAID второго типа. Что дает их использование?
9. Опишите способ создания оптических дисков CD-ROM.
10. Опишите систему кодирования данных, применяющуюся в оптических дисках.
11. Опишите физические принципы реализации оптических дисков CD-R.
12. Опишите физические принципы реализации оптических дисков CD-RW.
13. Поясните понятия «дорожка компакт-диска», «файл-образ», «сессия», «оглавление VTOC».
14. Чем отличаются диски DVD от компакт-дисков CD-ROM?
15. Какие мобильные носители информации вам известны?

Глава 13
Системный блок и периферийные устройства

Практически все обсуждавшиеся до сих пор устройства — микросхемы памяти, кэша и процессора, шины, дисководы жестких и сменных дисков — находятся внутри корпуса компьютера, который принято называть **системным блоком**. Как правило, устройства, служащие для организации ввода/вывода, находятся вне системного блока, и потому их часто называют **периферийными** устройствами.

13.1. Системный блок

Системный блок компьютера представляет собой корпус, коробку из металла, предназначенную для установки и крепления основных устройств компьютера.

В стандартном системном блоке персонального компьютера находятся:

- системная (материнская) плата;
- платы расширений, которые используются для подключения различных внешних устройств, например сетевая плата, видеоплата, звуковая плата и т. д.;
- один или несколько жестких дисков;
- дисководы гибких дисков, дисководы CD-ROM, CD-R, CD-RW, DVD;
- тактовый генератор;
- блок питания, система вентиляторов или кулеров (от cooler — охладитель), встроенный динамик.

Системной, или **материнской**, платой (рис. 13.1) называется основная плата компьютера, по которой проходят его основные шины и на которой размещены слоты и сокеты для крепления основных внутренних устройств компьютера — микросхем процессора, чипсета, кэша, оперативной памяти, а также плат расши-

рений. Платы расширений обычно содержат контроллер устройства и его буферную память. Они вставляются в слоты системной платы с помощью концевых разъемов, аналогичных концевым разъемам модуля памяти (см. рис. 7.2).

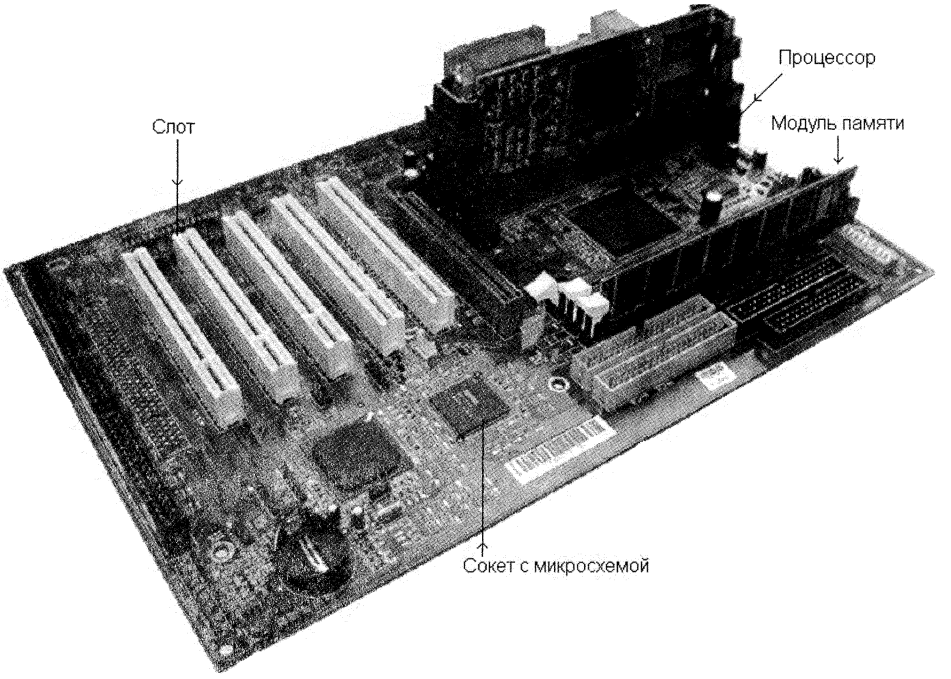


Рис. 13.1. Общий вид системной платы

Размеры системных блоков регулируются стандартами, первым из которых считается стандарт **AT** (от Advanced Technology — продвинутая технология), использовавшийся для корпусов персональных компьютеров IBM PC/AT. В настоящее время большинство корпусов компьютеров выпускается в соответствии со стандартом **ATX** (от AT eXtended — улучшенный AT), хотя уже разработано несколько новых стандартов.

Ранее широко использовались системные блоки горизонтального расположения с длиной корпуса около 35 см. Сейчас в основном выпускаются системные блоки вертикального расположения, имеющие несколько разновидностей:

- minitower (мини-башня) — высота 35 см;
- miditower (средняя башня) — высота 40 см;
- bigtower (большая башня) — высота 60 см;
- superbigtower (сверхбольшая башня) — высота от 1 м.

Корпуса последнего типа используются как стойки для размещения серверов.

На передней панели системного блока компьютера находятся кнопка включения электропитания Power и кнопка перезагрузки Reset. Там же находятся две

сигнальные лампочки: зеленая лампочка питания и желтая лампочка жесткого диска, — а также панели управления дисководами и выход встроенного динамика. У компьютеров, выпущенных в последние годы, на переднюю панель системного блока перенесен разъем шины USB.

На задней панели системного блока находятся: разъем для подключения электропитания, стандартные разъемы для подключения различных внешних устройств, выходы плат расширений, служащие для подключения звуковых устройств, видеоустройств (например, разъемы телевизионного входа/выхода TV In и TV Out), сетевых устройств и т. д.

В современных персональных компьютерах предусмотрены следующие разъемы, которые принято называть портами (см. 4.2.7):

- параллельный порт **LPT**, через который данные передаются байтами со скоростью около 2 Мбайт/с. Предназначен для подключения принтера, сканера, внешних дисководов и т. д.;
- последовательные порты **COM**, через которые данные передаются битами со скоростью около 100 Кбайт/с. Предназначены для подключения низкоскоростных внешних устройств, таких как мышь, клавиатура, модем и т. д.;
- последовательные порты **PS/2**, более современные, чем последовательный порт COM. Специально предназначены для подключения клавиатуры и мыши;
- порт **USB**, соответствующий унифицированному стандарту на шину со скоростью передачи данных до 12 Мбайт/с (у модификации 2.0 — до 60 Мбайт/с). К порту USB одновременно можно подключить до 127 внешних устройств. Предполагается, что со временем все устройства будут подключаться к разъемам типа USB;
- порт **FireWire** (или IEEE 1394), предназначенный для подключения высокоскоростных внешних устройств, таких как цифровые фото- и видеокамеры.

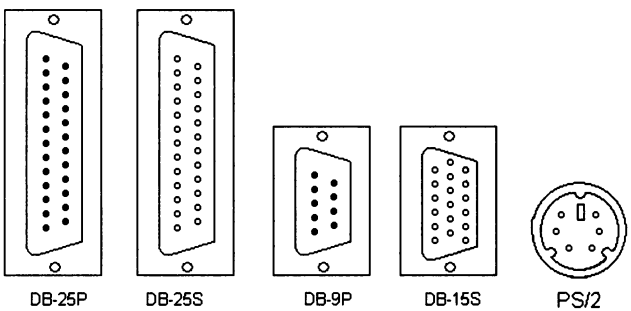


Рис. 13.2. Некоторые стандартные разъемы компьютера

В портах компьютера используются двух- и трехрядные разъемы типов S (розетка) и P (вилка) со стандартными геометрическими размерами. В частности, для LPT-порта используется двухрядный разъем DB-25S, который имеет 25 гнезд (рис. 13.2). Для COM-портов используется двухрядный разъем DB-25P, который

имеет 25 штырьков, или двухрядный девятиштырьковый разъем DB-9P. А для подключения дисплея используется содержащий 15 гнезд трехрядный разъем DB-15S. Компьютер имеет два одинаковых разъема типа PS/2 круглой формы, причем разъем и штекер клавиатуры окрашены в фиолетовый цвет, а разъем и штекер мыши — в зеленый. Все эти разъемы имеют стандартные форму и размеры, исключающие случайное неправильное соединение.

13.2. Дисплей и графическая подсистема

Одним из важнейших устройств компьютера, применяющихся для вывода информации, является дисплей или монитор (от monitor — устройство для слежения, контроля). На экран дисплея выводятся данные, вводимые с клавиатуры, результаты их обработки, а также всевозможная служебная информация.

Дисплеи бывают монохромные (то есть одноцветные — черно-белые, с желтым или зеленоватым оттенком) и цветные. Кроме того, различают **алфавитно-цифровые** и **графические** дисплеи. У алфавитно-цифровых дисплеев группа пикселов, занимающая небольшую прямоугольную область экрана и используемая для размещения изображения одного символа, образует **знакоместо**. Например, для растра размером 600 × 480 область, занимаемая знакоместом, образуется группой 8 × 8 пикселов. Изображение символа формируется примерно так же, как из группы точек на почтовом конверте получается изображение какой-либо цифры почтового индекса адресата. Подчеркнем, что у алфавитно-цифровых дисплеев не существует возможности работать с отдельным пикселом. Информация выводится на экран сразу целым знакоместом, символом. Поэтому такие дисплеи могут использоваться только для вывода различного рода текстов. Рисунки, графики, чертежи, картинки не могут быть выведены на алфавитно-цифровые дисплеи. В настоящее время алфавитно-цифровые дисплеи используются для управления различного рода серверами, то есть там, где отображение графики не является обязательным.

Графические дисплеи отличаются тем, что из программы можно управлять состоянием отдельного пиксела, и, следовательно, для них доступны все возможности формирования изображений.

Основными техническими характеристиками дисплеев являются:

- принцип действия;
- размер экрана по диагонали;
- разрешающая способность;
- размер «зерна» экрана;
- частота регенерации;
- форма экрана;
- класс защиты.

По принципу действия выделяют дисплеи на электронно-лучевой трубке (ЭЛТ, или CRT — от Cathode Ray Terminal, т. е. терминал на катодно-лучевой трубке), жидкокристаллические (ЖК, или LCD — от Liquid-Crystal Display, то есть жидкокристаллический дисплей) и плазменные дисплеи.

Принцип действия мониторов с электронно-лучевой трубкой в точности такой же, как у бытовых телевизоров. Электронная пушка, аналог катода в электронных лампах накаливания, вырабатывает луч — узконаправленный поток электронов, который с помощью системы отклоняющих пластин сканирует поверхность экрана дисплея. Точка пересечения луча с экраном представляет собой пиксел — элементарную единицу изображения. С помощью декодирующей схемы, на вход которой поступает закодированное изображение, пиксел переводится в одно из двух состояний — черное или белое: это позволяет формировать монохромные изображения. Для создания цветного изображения в мониторе устанавливаются три электронных пушки — красного, зеленого и голубого цвета. ЭЛТ-мониторы отличаются довольно большими габаритами, прекрасной цветопередачей и невысокой стоимостью.

Принцип действия жидкокристаллических дисплеев основан на свойствах жидких кристаллов, открытых еще в 1888 г. Они представляют собой вязкие органические молекулы, которые, с одной стороны, имеют структуру, аналогичную структуре кристалла, а с другой — ведут себя как молекулы жидкости. Оказалось, что оптические свойства жидких кристаллов зависят от ориентации молекул, а на ориентацию молекул жидкого кристалла можно воздействовать электрическим полем, что создает возможность для программно-управляемого построения изображения.

Экран LCD-дисплея состоит из двух стеклянных параллельных пластин, пространство между которыми заполнено жидкокристаллическим веществом. У жидкокристаллических дисплеев с **пассивной матрицей** на стеклянные пластины наносится сетка прозрачных электродов. Например, для обеспечения разрешающей способности экрана 800 × 600 сетка на задней пластине содержит 800 вертикальных проводов, а сетка на передней пластине — 600 горизонтальных. Источник света за задней пластиной освещает экран изнутри монитора. На провода сетки подается напряжение, которое различным образом ориентирует молекулы в разных точках экрана, определяя нужным образом цвет, яркость или контрастность в каждой его точке, в каждом пикселе. У жидкокристаллических дисплеев с **активной матрицей** вместо двух наборов сеток около каждого пиксела экрана находится крошечный элемент переключения напряжения электрического поля. Меняя соответствующим образом напряжение элемента в каждой точке, можно управлять изображением на экране.

Жидкокристаллические дисплеи отличаются малой толщиной и плоским экраном. Их стоимость пока выше, чем стоимость мониторов с электронно-лучевой трубкой. Причем мониторы с активной матрицей более качественные и более дорогие, а мониторы с пассивной матрицей имеют более бледное изображение, на них заметнее следы от смены кадров, но они и дешевле.

Самыми дорогими в настоящее время являются плазменные мониторы, которые обладают высоким качеством формируемого изображения и могут иметь значительные размеры — до 1 м и более по диагонали при толщине всего 10 см.

Перспективным направлением в развитии устройств отображения данных являются дисплеи, построенные по технологии **OLED** (от Organic Light Emitting Diodes — органические светодиоды). Во-первых, эти дисплеи не требуют дополнительной подсветки, так как вещество само испускает свет, а во-вторых, возможно размещение очень тонких экранов на гибкой основе.

Размер экрана дисплея по диагонали определяется в сантиметрах или дюймах. В настоящее время выпускаются мониторы с экранами от 9 до 42 дюймов или от 23 до 107 см. Наиболее распространенными являются экраны размером 15, 17, 19 и 21 дюйм. Для стандартных целей достаточно 17-дюймового экрана. При большом объеме работы с графикой желательно выбирать 19- или 21-дюймовые мониторы.

Важной характеристикой дисплеев является **разрешающая способность** экрана, определяющая *степень четкости* изображения. Разрешающая способность зависит от количества строк на весь экран и количества пикселов в строке. В настоящее время существует несколько стандартных разрешений, в частности: 800 × 600, 1024 × 768, 1152 × 864, 1280 × 1024, 1600 × 1200, 1600 × 1280, 1920 × 1200, 1920 × 1600, 2048 × 1536. Здесь первая цифра определяет количество пикселов в строке, а вторая — количество строк на экране. Возможное разрешение существенно зависит от фактического размера экрана. Например, для 17-дюймового монитора стандартным считается разрешение 1024 × 768, а максимальным может быть разрешение 1600 × 1200.

Отметим, что у ЭЛТ-мониторов разрешающая способность лучше, она может достигать 2048 × 1536, в то время как у лучших ЖК-мониторов она пока значительно ниже — до 1280 × 1024. Попутно заметим, что у телевизионных приемников наилучшим на сегодняшний день считается разрешение 1024 × 768.

Качество изображения определяется не только разрешающей способностью, но и так называемой **зернистостью** экрана. Зернистость разными производителями определяется либо как фактический линейный размер пиксела, либо как расстояние между двумя соседними пикселами. В настоящее время этот параметр у большинства мониторов равен 0,18–0,28 мм. Чем меньше размер зерна, тем лучше, но и дороже монитор.

Частота регенерации (обновления) — это параметр, который показывает, сколько раз в секунду обновляется изображение на экране дисплея. Без такого обновления невозможно формирование нормального зрительного восприятия телевизионного изображения, а также невозможна передача движений. Если частота регенерации меньше 60 Гц, то есть если обновление происходит менее чем 60 раз в секунду, то появляется мерцание изображения, что отрицательно сказывается на зрении. В настоящее время частота регенерации большинства мониторов составляет 60–100 Гц, а стандартной считается частота 85 Гц.

Экраны мониторов бывают выпуклые и плоские. В настоящее время большинство экранов, в том числе и у бытовых телевизоров, выпуклые. Вместе с тем

более перспективными моделями считаются мониторы с плоским экраном, например модель Trinitron, у которой экран абсолютно плоский по вертикали и лишь слегка искривлен по горизонтали.

С точки зрения техники безопасности работы с мониторами, необходимо учитывать **класс защиты** монитора, который определяется международными стандартами. В настоящее время действует стандарт под названием **TCO-2004**, выдвигающий самые жесткие требования к безопасному для человека уровню электромагнитных излучений, эргономическим и экологическим параметрам, а также к параметрам, определяющим качество изображения — яркости, контрастности, мерцанию, антибликовым и антистатическим свойствам покрытия экрана монитора.

Для создания изображения на экране дисплея необходим еще один компонент компьютера, который называют **видеоплатой**, **видеокартой** или **видеоадаптером**. Если быть точным, то это устройство следует называть графическим контроллером. Именно видеоадаптер определяет разрешающую способность монитора и количество передаваемых цветовых оттенков. Видеоадаптер вместе с дисплеем образуют **видеоподсистему** компьютера. В настоящее время в основном используются адаптеры типа **SVGA** (от Super Video Graphics Array — супервидеографический массив), способные передавать 16,7 млн. цветовых оттенков.

Для обеспечения такого количества цветов, а также хорошего разрешения видеоадаптеры содержат собственную **видеопамять** довольно большого объема — 64 Мбайт и выше. Построение высококачественных изображений и, тем более, какие-либо их преобразования, как правило, требуют выполнения большого количества математических операций. Чтобы освободить процессор компьютера от действий с изображениями и тем самым существенно ускорить их построение, а также повысить общую эффективность работы компьютера, современные видеоадаптеры берут на себя значительную часть этих операций. При этом часть работы по формированию изображения возлагается на аппаратные средства адаптера — микросхемы **видеоускорителя**, которые могут входить в состав видеоадаптера или размещаться на отдельной плате, подсоединяемой к адаптеру. Различают два типа видеоускорителей: плоские, или 2D (от 2-dimension — двухмерный), и трехмерные, или 3D (от 3-dimension — трехмерный). Требования современных видеоадаптеров, особенно с аппаратным ускорением, уже не удовлетворяются стандартными шинами компьютера. Поэтому для них были разработаны уже упоминавшиеся специализированные шины AGP.

13.3. Принтеры

Для получения результатов работы программ на бумаге в комплект персональных компьютеров включают печатающее устройство, которое обычно называют принтером (от print — печать). Основные технические характеристики принтеров:

- тип принтера — точечно-матричный, струйный или лазерный;
- возможность работы с цветом;

- ширина каретки;
- разрешение при печати;
- скорость печати;
- эксплуатационные расходы.

Существенной характеристикой любого принтера является **ширина каретки**, определяющая максимальные размеры, или **формат документа**, который может быть напечатан на данном принтере. Размер двух газетных разворотов, составляющий 840 × 1188 мм, принят в нашей стране как основа для стандартных форматов. Этот формат принято обозначать A0. Остальные форматы образуются из основного последовательным делением большего размера на два. Так, формату A1 соответствуют размеры 594 × 840 мм (две газетные страницы на одном листе бумаги), формату A2 — размеры 420 × 594 мм, формату A3 — 297 × 420 мм, A4 — 210 × 297 мм и т. д. до формата A10 с размерами 26 × 37 мм. В качестве стандартного размера листа бумаги для делопроизводства в нашей стране принят лист формата A4. Документы форматов A2 и A3 могут быть подготовлены только на принтерах с широкой кареткой. Документы формата A4 и меньше могут быть напечатаны и на принтере с узкой кареткой.

При печати графики важной характеристикой является **разрешение при печати**, которое измеряется в единицах **dpi** (от dots per inch — точки на дюйм). Один dpi равен одной точке, печатаемой на одном дюйме. Эта характеристика очень похожа на разрешающую способность мониторов, только речь идет о формировании изображения на бумаге, а не на экране монитора.

Скорость печати определяется как количество полностью отпечатанных листов или как количество печатаемых символов в единицу времени, обычно в минуту. Иногда скорость печати указывается как время, которое требуется для печати одного листа. При указании этой характеристики предполагается, что печатаемый материал не содержит графики.

Весьма важной характеристикой принтера является стоимость его эксплуатации, или **эксплуатационные расходы**. Современные принтеры обычно используют для выполнения печати специальное съемное устройство — **картридж** (от cartridge — патрон, заряд, катушка с пленками). Внутри картриджа размещается красящая лента или расходуемое при печати вещество: чернила или же специальный порошок, который принято называть **тонером** (от tone — тон, оттенок). Используемые при печати красящие ленты, чернила, тонеры, бумагу и т. д. называют **расходными материалами**. Эксплуатационные расходы включают в себя стоимость картриджа и стоимость заправки картриджа тонером или замены ленты в картридже, а также стоимость печати одного листа. Кроме того, в качестве характеристики экономичности указывается количество листов, которые можно напечатать на одной заправке картриджа.

По принципу действия используемые в настоящее время принтеры можно разделить на три группы — **точечно-матричные**, **струйные** и **лазерные**.

Печатающие головки точечно-матричных принтеров содержат матрицу — группу иголок, которые, выдвигаясь из головки принтера в определенных комбинациях

и ударяя по красящей ленте, оставляют на бумаге изображение символа. Принтеры этой группы могут печатать со скоростью (при черновой печати) от 180 до 400 символов в секунду, при этом на печать одного листа текста требуется от 9 до 20 с. Они могут печатать не только текст, но и графику, при этом характерным является разрешение 360 × 360 dpi. Некоторые модели обеспечивают печать в цвете, а также вывод сразу нескольких копий документа. Однако качество цветной печати невысокое. Кроме того, матричные принтеры довольно сильно шумят при печати. Точечно-матричные принтеры имеют сравнительно низкую собственную цену и невысокую стоимость расходных материалов — красящих лент, картриджей, бумаги.

Печатающие головки струйных принтеров имеют несколько форсунок, капиллярных сопел, через которые на бумагу выстреливаются очень маленькие капельки чернил разных цветов. Количество сопел может быть большим — 64 и выше. Таким образом, на бумаге получается цветное точечное изображение высокого качества. Струйные принтеры обладают хорошим качеством вывода цветных и графических изображений. В настоящее время разрешение при печати достигает 2400 × 1200 dpi. Однако скорость печати струйных принтеров относительно невысока — порядка 10 страниц в минуту. Принтеры этой группы довольно сложны в эксплуатации, так как требуют специального ухода за соплами. Засыхание чернил приводит к полному выходу из строя пишущего узла принтера. Вместе с тем стоимость самих струйных принтеров относительно невысока.

В лазерных принтерах используется практически такая же технология, как и в фотокопировальных устройствах. Главной частью лазерного принтера является вращающийся барабан или лента. Перед печатью листа на барабан подается напряжение порядка 1000 В. Луч лазера, несущий закодированную строку пикселов, проходит вдоль образующей барабана. Участки, на которые попадает луч, теряют свой электрический заряд — в результате получается строка с переменным потенциалом, отображающая очередную строку пикселов печатаемого текста. После формирования будущей строки пикселов барабан поворачивается на определенный угол и входит в контакт с резервуаром, содержащим тонер, который представляет собой красящий порошок, чувствительный к электростатическому полю. Тонер притягивается к заряженным точкам барабана. После еще одного поворота барабан с подготовленной строкой пикселов прижимается к бумаге, оставляя на ней изображение. Затем лист бумаги проходит через горячие валики, частицы тонера расплавляются и внедряются в бумажный лист, закрепляя сформированное изображение. После этого участок барабана, который несет напечатанную строку, разряжается от статического электричества и очищается от остатков тонера.

Для хранения кодов всех строк пикселов печатаемой страницы лазерные принтеры имеют собственную память, объем которой доходит до нескольких сотен мегабайт. Для лазерных принтеров характерны: высокая скорость печати — до 30 и более страниц в минуту, высокое, сравнимое с полиграфическим, качество печати цвета и графики при разрешающей способности до 2400 × 2400 dpi. Лазерные принтеры, особенно цветные, имеют относительно высокие собственную стоимость и стоимость эксплуатации.

ПРИМЕЧАНИЕ

Недавно компания IBM представила лазерный принтер Infoprint 4100, который печатает со скоростью 330 страниц в минуту. Такой принтер всего за две минуты способен отпечатать целый том романа Л. Н. Толстого «Война и мир». Принтер печатает документы на бумажных рулонах, которые затем разрезаются на страницы с помощью другого устройства.

13.4. Другие устройства компьютера

К основным устройствам ввода относится клавиатура, которая используется для первичного ввода данных в компьютер, а также для управления его работой. В настоящее время клавиатуры подавляющего большинства персональных компьютеров унифицированы и выполнены в стандартах 101/102-, 104/105- или 108-клавишных клавиатур. Вместе с тем формы и дизайн клавиатур довольно разнообразны. Можно отметить распространение в последнее время беспроводных клавиатур с инфракрасной передачей данных, которые не требуют подсоединения к системному блоку с помощью кабеля. Заметим, что клавиатуру вместе с дисплеем (а иногда и только клавиатуру) называют **консолью**.

К группе устройств ввода относится манипулятор «мышь», который обеспечивает удобный уровень управления работой программы и используется для ввода простейших видов графической информации — рисунков, чертежей и т. д. Мыши бывают механические, оптические, проводные и беспроводные (инфракрасные).

Устройство, которое называется **джойстик**, по принципу действия похоже на мышь. Вместо коробочки мыши, произвольно перемещаемой по поверхности стола, у джойстика имеется вертикальная рукоятка, один конец которой зафиксирован в шарнирном механизме, позволяющем произвольным образом наклонять и вращать ручку. Изменения в положении ручки джойстика соответствуют изменениям в положении корпуса мыши. Как и у мыши, у джойстика имеются кнопки, с помощью которых осуществляется управление программой. Существуют и специализированные разновидности джойстиков, предназначенные для имитации в игровых программах действий по управлению автомобилем или самолетом. В частности, они могут быть выполнены в виде руля автомобиля или штурвала самолета в комплекте с несколькими педалями.

Существует и еще одно устройство, принцип действия которого аналогичен принципу действия мыши. Это устройство называется **пенмаус**. Внешне пенмаус похож на шариковую ручку, на рабочем конце которой находится узел, регистрирующий ее перемещения.

В ноутбуках широко используется **трекбол**, представляющий собой стационарно устанавливаемый корпус, на верхней панели которого имеется шарик, приводимый в движение рукой. Можно считать, что трекбол — это перевернутая неподвижная мышь, шарик которой вращается пользователем непосредственно, а не при перемещении корпуса устройства.

Для ввода в память машины изображений, более сложных, чем простые рисунки, которые, как карандашом на листе бумаги, можно рисовать на экране мышью, разработаны **сканеры**. Сканер используется для передачи в память машины уже имеющихся на бумажном носителе чертежей, фотографий, иллюстраций и т. д. Оптическое устройство сканирует изображение, просматривая его узкими горизонтальными полосками, и сформированный сканером машинный код этого изображения передается в память.

Основными техническими характеристиками сканеров являются возможный формат сканируемого листа бумаги — A2, A3 или A4, и разрешающая способность, равная количеству точек, различаемых сканером на одном дюйме, и измеряемая в единицах dpi. Разрешающая способность сканеров в зависимости от модели изменяется в диапазоне от 300 до 4800 dpi. В некоторых случаях разрешения по горизонтали и вертикали у сканера отличаются друг от друга, тогда разрешающая способность указывается двумя значениями, например 300 × 600 dpi.

Для работы со сканером разработаны программы, позволяющие не только передавать в память машины изображение печатного или рукописного текста, но и распознавать этот текст. С их помощью можно не только факсимильно (от лат. fac simile — делай подобное), то есть точно, без изменений, воспроизводить такой текст на экране или на бумаге, но и работать с ним, как с текстом, набранным с помощью клавиатуры. Например, можно проверить его синтаксическую правильность, отредактировать или преобразовать в печатный документ.

Для ввода графической информации в память компьютера применяются также **дигитайзеры**, или графические планшеты. В основе их действия лежит фиксация положения специального пера относительно планшета или экрана дисплея. В последнем случае перо называется **световым**. Дигитайзеры могут быть использованы художниками для создания всевозможных рисунков, иллюстраций *без промежуточного нанесения* на бумагу или иной традиционный носитель. В отличие от сканера, который используется для передачи в память компьютера уже готового изображения, дигитайзеры служат для сохранения изображения в процессе его создания.

Возможности вывода графической информации на современных принтерах довольно велики. Несмотря на это существуют и специализированные устройства — **графопостроители**, или **плоттеры**, которые используются для подготовки на бумаге различного рода конструкторских документов, чертежей, графиков, рисунков. Они обеспечивают работу с цветом и документами очень больших форматов.

Для работы со звуком в мультимедийной среде к аппаратуре компьютера предъявляются определенные требования. Эти требования сформулированы в стандарте **MPC** (от Multimedia Personal Computer). В частности, в комплект машины должны входить: акустические стереоколонки, микрофон и **звуковой адаптер (звуковая плата, звуковая карта)**, который связывает различные акустические устройства с компьютером. Звук может быть введен как с микрофона,

так и с любого создающего звук устройства — магнитофона, проигрывателя, телевизора и т. д.

Среди менее распространенных периферийных устройств можно упомянуть синтезаторы звука, которые используются для воспроизведения звуков различных музыкальных инструментов, а также музыкальные приставки, применяемые профессиональными музыкантами для создания и аранжировки музыкальных произведений.

Среди перспективных периферийных устройств компьютера следует указать на **речевой ввод и вывод**. Они представляют собой устройства, распознающие голос человека и «понимающие» сказанные им фразы, а также устройства, синтезирующие человеческую речь.

Два класса устройств, **ограничители напряжения** и **источники бесперебойного питания**, используются для обеспечения надежного электропитания компьютеров. Устройства первой группы просто сглаживают резкие скачки напряжения в сети до приемлемого уровня, а устройства второй группы на некоторое время обеспечивают компьютер автономным электропитанием при его внезапном отключении и позволяют в это время сохранить результаты текущей работы.

13.5. Компактная условная формула — характеристика компьютера

В стандартный комплект персонального компьютера при его продаже обычно входят системный блок, клавиатура, мышь и монитор. При необходимости отдельно приобретаются принтер, сканер, модем и другие периферийные устройства. В настоящее время для описания продаваемого компьютера используется краткая условная формула, дающая довольно полное представление об основных технических характеристиках машины. В эту формулу входят:

1) модель процессора, его тактовая частота, наличие и объем кэша;
2) объем и тип оперативной памяти;
3) используемые шины;
4) объем жесткого диска;
5) наличие флоппи-дисковода;
6) наличие и тип дисковода компакт-дисков;
7) наличие видеоплаты, ее тип и объем дополнительной памяти;
8) наличие звуковой платы, ее тип;
9) стандарт клавиатуры и наличие в комплекте мыши.

Технические характеристики устройств компьютера обычно перечисляются именно в таком порядке. Если какая-либо характеристика не указывается, то ее отсутствие не влияет на порядок задания остальных параметров. Возьмем, например, следующую «формулу» компьютера:

P4–2400, 512 L2/256 DIMM DDR/PCI, USB/80 Gb/FDD 3,5''/CD-RW52x/AGP4x 32Mb/SB AUDIGY 5.1/Mouse/Keyboard 108

Формула всегда начинается с указания типа процессора и его тактовой частоты. В данном случае P4–2400 означает, что основу компьютера составляет процессор типа Pentium 4 с тактовой частотой 2400 МГц, или 2,4 ГГц, и внешним кэшем объемом 512 Мбайт (512 L2). Далее указываются объем и тип оперативной памяти: 256 DIMM DDR означает двухрядный модуль (DIMM) оперативной памяти типа DDR SDRAM объемом 256 Мбайт. Используемые в компьютере специализированные шины указываются не всегда. В нашем примере они указаны — это шины типа PCI и USB. Зато объем жесткого диска указывается всегда — это очень важный показатель. В рассматриваемом примере указано, что объем жесткого диска равен 80 Гбайт (80 Gb). Иногда добавляются тип интерфейса жесткого диска (EIDE или SCSI) и скорость его вращения (например, 7200 об./мин). Далее в формуле показано наличие в комплекте дисковода гибких дисков (FDD 3,5") и дисковода оптических дисков с возможностью многократной перезаписи (CD-RW) и 52-кратной скоростью обмена (52x). В состав системного блока также входит видеоплата типа AGP со скоростью обмена 1,06 Гбайт/с (4x) и дополнительной памятью объемом 32 Мбайт. Имеется звуковая плата модели SB (Sound Blaster) AUDIGY 5.1. Завершается формула указанием на наличие в комплекте поставки мыши (Mouse) и 108-клавишной клавиатуры (Keyboard 108). Заметим, что наличие двух последних компонентов обычно подразумевается и в формуле специально не оговаривается.

Монитор характеризуют отдельно указанием модели, типа, размера экрана, размера зерна, разрешения и частоты, а также наличием сертификата класса защиты, например: Samsung 763MB/CRT/17"/0,20/1280×1024@85/TCO99. В данном случае речь идет о мониторе на электронно-лучевой трубке (CRT) модели Samsung 763MB с 17-дюймовым экраном (17"), размером зерна 0,20 мм, разрешением 1280 × 1024, частотой регенерации 85 Гц (@85) и наличием сертификата класса защиты TCO-99.

Выбирая компьютер, необходимо обращать внимание на то, чтобы технические характеристики устройств были *сбалансированы*. Бессмысленно устанавливать мощный процессор на системную плату, шина памяти которой имеет низкую пропускную способность, с малым объемом оперативной памяти и без внешнего кэша. Не менее важную роль в общей производительности компьютера играют возможности чипсета и дисковой подсистемы.

Дополнительный материал по обсуждавшимся в главе вопросам можно найти в изданиях [30], [34], [35].

Контрольные вопросы и упражнения

1. Перечислите состав устройств, находящихся внутри системного блока.
2. Какие стандартные разъемы имеются у персонального компьютера?
3. Охарактеризуйте роль системной платы в конструкции компьютера.
4. Перечислите технические характеристики дисплеев.
5. Опишите принцип действия дисплеев на электронно-лучевой трубке.
6. Опишите принцип действия жидкокристаллических дисплеев.
7. Дайте сравнительную характеристику различных типов дисплеев.
8. Чем отличаются дисплеи с активной матрицей от дисплеев с пассивной матрицей?
9. Чем отличается технология OLED?
10. Поясните понятия «разрешающая способность», «зернистость экрана», «частота регенерации», «класс защиты».
11. Что входит в видеоподсистему компьютера?
12. Какую роль играет видеоадаптер?
13. Для чего нужен видеоускоритель?
14. Перечислите технические характеристики принтеров.
15. Опишите принцип действия точечно-матричных принтеров.
16. Опишите принцип действия струйных принтеров.
17. Опишите принцип действия лазерных принтеров.
18. Дайте сравнительную характеристику принтеров различных типов.
19. Охарактеризуйте известные вам периферийные устройства компьютера.
20. Опишите структуру краткой формулы-характеристики аппаратного состава компьютера.
21. Опишите структуру краткой формулы-характеристики мониторов.

Глава 14
Оценка производительности вычислительных систем

Ранее в качестве приближенной меры производительности (мощности) вычислительных систем мы использовали две важные характеристики: тактовую частоту центрального процессора и объем оперативной памяти. Подобный подход неплохо работал в вычислительных машинах, в которых не было многоуровневого кэша, конвейеров, «хитрых» методов улучшения загрузки конвейера, суперскалярных процессоров и т. д. Но дальнейшее развитие архитектуры компьютера показало, что ситуация с измерением мощности компьютера далеко не так проста.

Проиллюстрируем это замечание наглядным примером, взятым из [11]. В 5.1 приведены данные машины EDSAC, считающейся первой машиной первого поколения (год выпуска 1949): тактовая частота 0,5 МГц, производительность 100 арифметических операций в секунду. Центральные процессоры вычислительной системы Hewlett-Packard Superdome в 2002 г. работали на частоте 770 МГц, а ее производительность оценивалась в 192 млрд. арифметических операций в секунду. То есть тактовая частота выросла «всего» в 1540 раз, в то время как производительность увеличилась почти в 2 миллиарда раз. «Дополнительный» прирост обеспечен не улучшением физических характеристик процессора и других центральных устройств компьютера, а широчайшим внедрением параллелизма и сопутствующих архитектурных решений, развитием математических и алгоритмических методов, а также соответствующего программного обеспечения.

Тщательный анализ показывает, что производительность вычислительной системы зависит от множества факторов, в том числе:

- скоростных характеристик и разрядностей системы шин компьютера;
- скоростных характеристик и объема внешних запоминающих устройств, особенно магнитных дисков;
- устройств, обеспечивающих обмен данными между входящими в вычислительную систему процессорами;

- возможностей используемой операционной системы, ее «умения распорядиться» возможностями аппаратуры, а особенно того, как организована параллельная работа центральных процессоров;
- «умения» трансляторов подготовить машинный код программы к исполнению в параллельной среде — на нескольких функциональных блоках, конвейерах, процессорах и т. д.;
- возможностей организации параллельного исполнения программы, имеющихся в используемых языках программирования;
- мощности применяемых алгоритмических и математических методов, того, насколько удачным оказался выбранный способ распараллеливания задачи, то есть способ выделения участков, предназначенных для одновременного, параллельного исполнения на нескольких процессорах или компьютерах вычислительной системы;
- степени соответствия имеющихся аппаратных средств и выбранного способа распараллеливания;
- и наконец, совсем уже плохо контролируемого фактора — от возможностей распараллеливания, которые заложены в «природу» самой решаемой задачи.

В связи с наличием такого огромного количества факторов, влияющих на производительность вычислительной системы, и настоятельной необходимостью каким-то образом все-таки ее оценивать, в настоящее время используется несколько способов указания мощности компьютеров. Если отбросить детали, таких способов всего три:

- оценка с помощью тактовой частоты;
- оценка с помощью указания количества операций, выполняемых в единицу времени;
- тестирование на специально отобранных программах.

14.1. Оценка производительности тактовой частотой

Тактовая частота используется как характеристика процессора в тех случаях, когда требуется только приблизительная оценка мощности, например для описания персональных компьютеров, применяющихся для решения офисных задач, для развлечений и в других целях. Чем выше тактовая частота, тем быстрее выполняются команды, тем большее количество команд в единицу времени выполняет процессор, тем выше его производительность. Применение тактовой частоты для оценки мощности облегчается тем, что это довольно легко измеряемый и воспринимаемый параметр. В случае многопроцессорных систем тактовая частота используется как дополнительная характеристика отдельного процессора, входящего в систему.

Использовать тактовую частоту для реального представления о производительности компьютера сложно, так как нужно дополнительно знать множество факторов, например среднее количество тактов, приходящихся на одну машинную команду, количество ступеней конвейера, количество функциональных блоков в суперскалярном процессоре, параметры всех уровней кэша и т. д. Одновременный учет всех этих факторов — довольно сложная задача. Особенно слабое представление дает тактовая частота о производительности многопроцессорных вычислительных систем.

14.2. Пиковая и реальная производительность

Для многих способов оценки различают *пиковую* (от peak — высшая точка) и *реальную* производительность вычислительной системы. Пиковая производительность представляет собой полученную теоретическим путем верхнюю оценку мощности вычислительной системы, а реальная производительность определяется экспериментальным путем, во время выполнения реальных программ. Пиковую производительность рассчитывают в предположении, что при выполнении программы все устройства компьютера работают на максимальном уровне своих возможностей. К пиковой производительности можно подойти довольно близко, но достичь ее в реальных условиях невозможно. Пиковая производительность для любой вычислительной системы рассчитывается однозначно, однако она слабо связана с конкретными показателями, которые могут быть достигнуты для конкретных задач: одних задачах это может быть 90 % процентов от пиковой, а на других — только 5–10 %.

14.3. Единицы MIPS

Для более точной характеристики мощности вычислительных систем используется подход, основанный на указании количества машинных команд, выполняемых системой в единицу времени. Отметим, что эту характеристику можно использовать и для оценки производительности многопроцессорных машин, если учитывать количество команд, выполняемых всей системой.

В этом подходе оценка производительности вычислительных систем производится в **единицах MIPS** (от Million Instructions Per Second — миллион машинных команд в секунду), в которых мощность компьютера равна отношению количества выполненных машинных команд (инструкций) ко времени их выполнения. Отличие этого способа оценки производительности в том, что в расчетах не различают формат данных, над которыми выполняет действие центральный процессор, то есть используется реальная смесь команд программы, состоящая из действий и над целочисленными, и над вещественными данными. Очевидное удобство этого способа — в его простоте и интуитивной понятности.

Основной недостаток использования единиц MIPS — в том, что результат зависит от системы команд процессора. Поэтому сложно сравнивать производитель-

ности компьютеров с разными системами машинных команд. Кроме того, известно, что различные команды выполняются процессором за разное время, а разные программы содержат «быстрые» и «медленные» команды в различных пропорциях. Следовательно, при выполнении на одном и том же компьютере разных программ можно получить разные оценки его производительности, что также препятствует широкому использованию этого показателя.

14.4. Единицы Flops

Альтернативной единицей измерения производительности вычислительных систем являются **флопы**, или **единицы Flops** (от Floating point operation per second — операции с плавающей точкой в секунду). В этом случае производительность системы равна отношению количества операций над вещественными данными (в формате с плавающей точкой) ко времени их выполнения. В современных условиях более часто используются кратные единицы: мегафлопы (1 Mflops = 10^6 Flops), гигафлопы (1 Gflops = 10^9 Flops), терафлопы (1 Tflops = 10^{12} Flops).

Эта единица измерения отличается от предыдущей двумя особенностями. Во-первых, при измерении в единицах Flops учитываются только операции над вещественными данными, а во-вторых, в оценке участвуют не машинные команды процессора, а выполненные операции над вещественными числами. Разница в том, что одна операция над вещественными числами (например, умножение или извлечение квадратного корня) может быть задана различными последовательностями машинных команд. Количество операций над вещественными числами зависит только от решаемой задачи и не зависит от реализующей вычисления машинной программы. Поэтому измерение в единицах Flops более объективно отражает производительность компьютера.

К сожалению, вне операций над вещественными данными эта система оценки производительности неприменима, так как для программ, слабо, использующих или вообще не использующих вычисления с вещественными данными (например, для программ-компиляторов), показатель производительности в единицах Flops оказывается очень малым.

У этого способа, так же как и у предыдущего, имеется недостаток, проявляющийся в существенной зависимости производительности системы от выполняемой программы. Как и в предыдущем случае, это объясняется различным соотношением между «быстрыми» и « медленными» операциями, но теперь уже не в программе, а в решаемой задаче. Кроме того, для программ с короткими циклами, когда все команды цикла могут одновременно находиться в кэше, производительность машины оказывается выше, чем для программы с циклами, в которых приходится обращаться к оперативной памяти. А для программ, в которых можно организовать много параллельных ветвей, например для программ, работающих с матрицами, производительность многопроцессорной системы окажется существенно выше, чем ее же производительность во время выполнения программы, не допускающей распараллеливания.

14.5. Тесты LINPACK

Из-за отмеченных недостатков единиц MIPS и Flops для сравнения производительности компьютеров было предложено использовать в качестве критерия время выполнения специально подобранной эталонной программы или же связанные с этим временем показатели. Программы, на которых осуществляется тестирование, иногда называют *бенчмарками* (от bench-mark — отметка уровня). К настоящему времени создано довольно много различных тестовых и эталонных программ.

Одной из наиболее известных систем оценки является тест LINPACK, представляющий собой пакет программ на языке Фортран, предназначенных для решения систем линейных алгебраических уравнений больших размерностей (до нескольких миллионов неизвестных) с плотной матрицей методом Гаусса с выбором главного элемента. Имеется несколько разновидностей этого теста, например LINPACK TPP (от Toward Peak Performance — направляющийся к пиковой производительности) и HPL (от High-Performance LINPACK — высокопроизводительный LINPACK).

List for June 2005

R_{max} and R_{peak} values are in GFlops. For more details about other fields, please click on the button "Explanation of the Fields"

DETAILS EXPLANATION OF THE FIELDS

1-100 | 101-200 | 201-300 | 301-400 | 401-500

Rank	Site Country/Year	Computer / Processors Manufacturer	R_{max} R_{peak}
1	DOE/NNSA/LLNL United States/2005	*BlueGene/L* **eServer Blue Gene Solution** / 65536 IBM	136800 183500
2	IBM Thomas J. Watson Research Center United States/2005	*BGW* **eServer Blue Gene Solution** / 40960 IBM	91290 114688
3	NASA/Ames Research Center/NAS United States/2004	*Columbia* **SGI Altix 1.5 GHz, Voltaire Infiniband** / 10160 SGI	51870 60960
4	The Earth Simulator Center Japan/2002	**Earth-Simulator** / 5120 NEC	35860 40960

Рис. 14.1. Фрагмент списка Top500 по состоянию на июнь 2005 г.

Для проведения тестирования формируется некоторая система линейных уравнений максимально возможной для имеющегося объема размерности и измеряется время ее решения на тестируемой вычислительной системе. Количество операций с вещественной точкой K, которые должны быть выполнены для получения

решения, равно $K = 2n^3/3 + 2n^2$, оно однозначно зависит от заданной размерности матрицы n, поэтому дальнейшая оценка производительности в единицах Flops не вызывает затруднений.

В настоящее время тесты LINPACK используются для определения списка Top500 — пятисот самых мощных вычислительных систем мира. Этот список можно найти в Интернете по адресу http://www.top500.org. На рис. 14.1 приведен фрагмент списка по состоянию на июнь 2005 г. В нем указываются занятое место (Rank), страна и год разработки (Country/Year), название компьютера и производитель процессора (Computer/Processor Manufacturer), максимальная и пиковая производительность в гигафлопах.

14.6. Ливерморские циклы

В тестировании вычислительных систем по методике LINPACK происходит проверка скоростных характеристик системы только на одном (и при этом очень узком) классе задач. В реальности встречается несравненно больше разнообразных задач, в том числе и вычислительного характера. Чтобы выявить возможности компьютера по решению задач других классов, стали применять тестирование на реальных программах, которые используют различные вычислительные методы. Одна из таких систем основана на измерении производительности с помощью так называемых *ливерморских циклов*, представляющих собой тщательно отобранные фрагменты программ на языке Фортран, которые эксплуатируются в Ливерморской национальной лаборатории (США).

По этой методике тестирование осуществляется на наборе из 24 операторов цикла, которые являются основными, наиболее существенными частями программ, реализующих часто встречающиеся численные методы решения вычислительных задач гидродинамики, ядерной физики и т. д. Заметим, что в связи с использованием основных частей (ядер) программ обсуждаемая система известна еще и как тест *LFK* (от Livermore Fortran Kernels — ливерморские фортрановские ядра).

Ливерморские циклы дают более точную картину по сравнению с тестом LINPACK, поскольку в тестировании используются не одна программа, реализующая единственный вычислительный метод, а целая группа программ, реализующих несколько методов. Вместе с тем тестирование все еще осуществляется на программах из одной и той же проблемной области, которые хотя и являются весьма важным классом приложений, но все-таки имеют схожую специфику.

14.7. SPEC и другие тесты

В настоящее время при оценке производительности в основном персональных компьютеров большой популярностью пользуется целое семейство тестов, созданных некоммерческой специализированной корпорацией **SPEC** (от Standard Performance Evaluation Corporation — корпорация стандартов оценки производительности).

В основу этих тестов положены реально используемые программы из самых разных областей применения информационных технологий.

Исходный вариант, относящийся к 1992 г., содержал две группы тестов. Группа с названием CINT92 состоит из шести программ на языке C, которые обеспечивают решение задач по теории цепей, разработку логических вентильных схем, упаковку текстовых файлов, включают интерпретатор для языка ЛИСП и т. д. Эта группа программ служит для оценки производительности систем с точки зрения операций над целочисленными данными. Вторая группа тестов с названием CFP92 включает 12 программ на языке C и две — на Фортране. Эти программы, обеспечивающие моделирование методом Монте-Карло, составление прогноза погоды и т. д., служат для оценки производительности систем с точки зрения операций над вещественными данными.

Результатом тестирования считаются отношения времен выполнения по каждой тестовой программе на испытуемой машине к временам их выполнения на эталонной машине. В качестве эталона выбрана вычислительная система VAX 11/780. Из результатов отдельных тестов формируются две интегральные оценки: SPECint92, равная среднему геометрическому оценок, полученных в индивидуальных тестах по группе CINT92, и SPECfp92, равная среднему геометрическому оценок, полученных в индивидуальных тестах по группе CFP92. Таким образом, в тестах SPEC оценки не измеряются в единицах MIPS или Flops, а являются безразмерными относительными величинами, показывающими, во сколько раз быстрее работает испытуемая машина по отношению к эталонной.

Аналогичным образом построены и более поздние варианты данных тестов и интегральных оценок SPECint95 и SPECfp95, SPECint2000 и SPECfp2000 и т. д., а также другие специализированные тесты SPEC. Можно упомянуть, например, тест SPEChpc96, обеспечивающий оценку мощности вычислительных систем, состоящих из нескольких десятков процессоров, а также тест SPEC OMPL2001, который может применяться для тестирования систем, содержащих до 512 процессоров. В систему SPEC входят тесты SPECjbb и SPECweb, служащие для тестирования различных разновидностей серверов, а также некоторые другие. Корпорация SPEC постоянно работает над созданием новых тестовых систем, улучшением и обновлением ранее выпущенных. Это, например, популярные тесты, использующие реальные, широко распространенные приложения SPEC for Maya 6, SPEC for 3ds max 6, SPEC for SolidWorks 2003, SPEC viewperf и т. д.

Кроме тестов SPEC в последние годы появилось еще несколько систем тестирования, созданных некоммерческими организациями. В основном эти системы ориентированы на приложения баз данных и другие невычислительные классы приложений. Можно упомянуть, скажем, системы тестирования TPC-A, TPC-B, TPC-C Совета по оценке производительности обработки транзакций TPC (от Transaction Processing Performance Council) и большой набор тестов SAP (от Standard Application) Benchmark.

В последнее время популярность приобрели комплексные методики тестирования производительности компьютеров, основанные на комплексах программ из различных областей применения. В частности, в тестовый комплекс включают программы архивации, моделирования физических процессов, растровой и трехмерной

графики, автоматизации проектирования, кодирования мультимедийных данных, игровые и некоторые другие программы. В комплексы часто включают программы 7-zip, WinRAR, CPU RightMark, Adobe Photoshop, 3DMark, PC Mark, WebMark, VeriTest Business Winstone, VeriTest Multimedia, Content Creation Winstone, SiSoftware Sandra, Adobe Acrobat Distiller, ABBYY Fine Reader, DOOM. Следует отметить, что набор средств, используемых для комплексной оценки производительности вычислительных систем, постоянно изменяется и довольно сильно зависит от массовых предпочтений, установившихся в период тестирования.

Оценивая производительность компьютера по любой системе тестирования, необходимо иметь в виду, что на разных тестах вычислительные системы дают разные показатели производительности. На одних тестах одна архитектура может выигрывать у другой, а на других — проигрывать. *Проблема общепризнанной, удобной, адекватной оценки мощности вычислительных систем до сих пор не имеет удовлетворительного решения.*

Дополнительный материал по обсуждавшимся в главе вопросам можно найти в изданиях [11], [6].

Контрольные вопросы и упражнения

1. Перечислите и охарактеризуйте факторы, от которых зависит производительность вычислительных систем.
2. Перечислите и охарактеризуйте базовые способы оценки производительности вычислительных систем.
3. Опишите способ оценки производительности с помощью тактовой частоты. Охарактеризуйте его достоинства и недостатки.
4. Как определяется пиковая производительность вычислительных систем? Чем она отличается от реальной?
5. Опишите способ оценки производительности в единицах MIPS. Охарактеризуйте его достоинства и недостатки.
6. Опишите способ оценки производительности в единицах Flops. Охарактеризуйте его достоинства и недостатки.
7. Дайте сравнительную характеристику способов оценки производительности вычислительных систем.
8. Какие системы тестирования производительности вам известны?
9. Опишите способ оценки производительности вычислительных систем с помощью тестов LINPAC. Охарактеризуйте его достоинства и недостатки.
10. Опишите способ оценки производительности вычислительных систем с помощью ливерморских циклов. Охарактеризуйте его достоинства и недостатки.
11. Опишите способ оценки производительности вычислительных систем с помощью тестовой системы SPEC. Охарактеризуйте его достоинства и недостатки.
12. Охарактеризуйте комплексную систему тестирования.

Глава 15
Классификация архитектур

К настоящему времени в мире разработаны тысячи различных моделей компьютеров, относящихся к самым разным архитектурам. Эти модели отличаются друг от друга устройством, способами кодирования информации, системами команд, объемом запоминающих устройств, скоростью обработки данных и т. д. Чтобы ориентироваться в этом многообразии вычислительных систем, применяются различные классификационные схемы, основанные на разных признаках.

Перед обсуждением классификации архитектур следует отметить, что многие из существующих классификационных схем, во-первых, довольно условны и не являются общепринятыми, а во-вторых, с течением времени претерпевают определенные изменения, связанные с бурным развитием информационных технологий и невозможностью точно предсказать направления будущего развития компьютерной техники. Поэтому мы рассмотрим только наиболее распространенные в настоящее время классификационные схемы:

- по принципу действия;
- по поколениям;
- по областям применения;
- по архитектуре набора команд;
- по разрядности машинного слова;
- по типу управления;
- по степени и способу организации параллелизма.

Используя любую схему классификации, необходимо учитывать, что имеются компьютеры и группы компьютеров, промежуточных по своему положению, то есть по одним признакам попадающих в одну классификационную категорию, а по другим — в другую.

15.1. Классификация по принципу действия

Классифицирующим признаком этой схемы являются тип обрабатываемых сигналов и соответствующий этому типу принцип работы вычислительной системы. Как было выяснено в главе 1, существуют дискретные и непрерывные сигналы и сообщения. В связи с этим вычислительные машины делятся на два основных класса:

- дискретные, или цифровые, вычислительные машины (**ЦВМ**), которые применяются для обработки дискретных сигналов и сообщений;
- аналоговые вычислительные машины (**АВМ**), используемые для обработки непрерывных сигналов и сообщений.

Все обсуждавшиеся ранее вычислительные системы относятся к группе цифровых машин. Основной отличительной особенностью их принципа действия является интерпретация любых действий над дискретными сигналами как арифметических (сложение, умножение и т. д.) или логических (конъюнкция, дизъюнкция и т. д.) операций над двоичными кодами. С помощью математических и алгоритмических методов и приемов необходимая обработка данных сводится к некоторой последовательности таких операций. Результатом обработки является двоичный код, который затем отображается в текстовом, числовом или графическом виде. Достоинствами цифрового подхода являются универсальный характер, возможность получения результата практически с любой необходимой точностью, наглядность, удобство работы. Отметим, что именно цифровые машины обычно имеются в виду при использовании термина «компьютер».

В аналоговых вычислительных системах используется не расчет по некоторому алгоритму, а моделирование, то есть построение реальной физической модели, адекватной решаемой задаче. Явления, протекающие в модели, оказываются подобны изучаемым явлениям, если модель и решаемая задача описываются одними и теми же системами математических уравнений.

В аналоговых машинах решаемая задача обычно моделируется электрическими схемами, и решение получается после подачи на входы схемы напряжений, соответствующих исходным данным задачи. Решение получается не путем последовательного выполнения арифметических или других действий, которое может занимать секунды, часы и дни, а сразу же после стабилизации уровней напряжения во всех участках цепи. При этом используются не два дискретных уровня напряжения, интерпретируемых как цифры 0 и 1, а целый интервал значений напряжений, каждая точка которого интерпретируется как вещественное значение. Изменение с течением времени напряжения в некоторой точке цепи соответствует изменению значения некоторой вещественной переменной моделируемой задачи. Решением считается совокупность значений параметров (ток, напряжение) моделирующей электрической схемы во всех ее точках.

Поскольку решение получается одновременно во всех точках цепи, можно сказать, что в аналоговой машине имеет место максимально возможная для решаемой задачи степень параллелизма, которая определяется не аппаратными возможностями машины, а самой структурой решаемой задачи.

Большинство аналоговых вычислительных машин имеют фиксированную (неизменяемую) или коммутируемую архитектуру, подобную архитектуре машины «Марк 1». Основу машины составляют соединяемые между собой электрическими кабелями функциональные блоки, которые выполняют над непрерывными сигналами такие операции, как дифференцирование, интегрирование и т. д. При любом изменении программы приходится выполнять перекоммутацию соединений между функциональными блоками, на что требуется довольно много времени.

С точки зрения решения задачи, можно считать, что каждый функциональный блок получает на входе функцию и выдает на выходе ее производную, ее первообразную или какую-либо другую функцию-результат. При этом нет необходимости использовать для вычисления, например, определенного интеграла некоторую приближенную формулу, скажем, формулу Симпсона. Функции-результаты регистрируются, как правило, на осциллографах.

В связи с высоким параллелизмом производительность аналоговых машин на специальных классах задач может многократно превосходить производительность самых современных цифровых машин. Но при этом точность результатов невысока, всего 1–6 %, что соответствует обычному уровню точности в физических экспериментах и инженерных задачах.

Цифровые вычислительные машины могут использоваться для решения задач любых классов, в то время как аналоговые машины, вообще говоря, относятся к узкоспециализированным системам, которые могут успешно применяться только при решении задач нескольких специальных классов. Наиболее выгодно использование аналоговых машин для решения задач, описываемых системами дифференциальных, интегральных и интегрально-дифференциальных уравнений, например задачи Коши. Широко применяются аналоговые машины в системах автоматического регулирования, а также в бортовых вычислительных и управляющих системах на судах, самолетах, ракетах.

В некоторых аналоговых машинах для моделирования используются не уровни напряжений в электрической цепи, а электрические процессы в сплошной объемной проводящей среде, например такой как электролит. Это позволяет моделировать решение сложных задач гидродинамики, теплопроводности и т. д., описываемых уравнениями с частными производными, такими как уравнение Лапласа.

В 1940—1960-е гг. аналоговые вычислительные машины были почти так же широко распространены, как и цифровые. Затем на фоне огромных достижений цифровой техники аналоговые машины отошли на задний план. В настоящее время наблюдается возрождение интереса к этим системам. Поскольку и аналоговые, и цифровые машины обладают как определенными достоинствами, так и недостатками, большой интерес вызывают так называемые *гибридные* вычислительные машины (**ГВМ**), которые сочетают в себе достоинства аналоговых и цифровых машин.

В этом учебнике рассматривается только архитектура *цифровых вычислительных машин*. Более подробное обсуждение аналоговых машин можно найти, например, в [2], [22].

15.2. Классификация по поколениям

Классификация вычислительных систем по поколениям соответствует историческому процессу развития вычислительной техники. Основными классифицирующими признаками являются элементная база и соответствующие ей технические параметры вычислительных систем: быстродействие процессора, объем оперативной и внешней памяти, надежность, стоимость и т. д. В качестве дополнительных факторов во внимание принимаются характер программного обеспечения, преимущественная сфера использования и некоторые другие. По совокупности этих признаков к настоящему времени принято выделять пять поколений вычислительной техники.

В 5.1 довольно подробно прослеживается процесс развития средств вычислений вплоть до первого поколения компьютеров. К первому поколению относятся машины, созданные в период, начинающийся с компьютера EDSAC и заканчивающийся примерно в конце 1950-х гг. Их элементной базой были электронные лампы накаливания (см. рис. 3.18, *а*). Оперативная память машин создавалась на ферритовых сердечниках. Ее объем составлял несколько тысяч чисел, а среднее быстродействие процессора было на уровне 5–30 тыс. арифметических операций в секунду. Программировать приходилось в машинных кодах, немного позднее на автокоде или Ассемблере. Создавались машины первого поколения в единичных экземплярах и использовались в основном для выполнения математических вычислений в военных, технических и научных целях.

Эти машины стоили очень дорого, занимали огромные площади, были не совсем надежны в работе, имели маленькую скорость обработки информации и могли хранить очень мало данных. Работать с этими машинами могли только профессионалы высочайшей квалификации. Чтобы обслуживать машину, ремонтировать ее при сбоях и т. д., требовался целый коллектив специалистов. Несмотря на все перечисленные недостатки, их применение для обработки информации оказалось весьма успешным.

В качестве типичных примеров машин первого поколения можно указать американские компьютеры UNIVAC, IBM-701, IBM-704, а также советские машины БЭСМ и М-20.

Ко второму поколению относят машины, элементной базой которых является транзистор (см. рис. 3.18, *б*), изобретенный в 1951 г. Уильямом Шокли. Машины второго поколения создавались в период с конца 50-х и до середины 60-х гг. XX в. У этих машин значительно уменьшились стоимость и габариты, выросли надежность, скорость работы и объем хранимой информации. Типичный объем оперативной памяти составлял 64–128 Кбайт, а скорость обработки данных возросла до 1 млн. операций в секунду.

В архитектуре машин второго поколения было введено совмещение во времени работы центрального процессора и каналов ввода/вывода, а также появился связанный с совмещением многопрограммный режим. Были созданы машины с распараллеливанием выборки команд и данных из оперативной памяти.

С появлением специальных алгоритмических языков, таких как Фортран и Алгол-60, а также библиотек стандартных программ существенно упростилось применение машин для решения практических задач в различных областях. Машины стали использовать для стандартных инженерных расчетов, в экономической деятельности, для оптимизации работы отдельных предприятий и даже отраслей и во многих других областях.

Типичными машинами второго поколения в нашей стране были БЭСМ-4, М-220, Минск-22, БЭСМ-6. Причем машина БЭСМ-6 в этот период считалась одним из лучших в Европе компьютеров.

Машины третьего поколения выполнены на интегральных схемах (см. рис. 3.18, *в*). Отметим, что первая интегральная схема была создана в 1959 г. Д. Килби. Переход от транзисторов к интегральным схемам вызвал соответствующие изменения в стоимости, размерах и надежности машин. Емкость оперативной памяти увеличилась до мегабайта, а скорость — до десятков миллионов операций в секунду.

Важнейшим отличием третьего поколения компьютеров стал выпуск машин *семействами*.

ВНИМАНИЕ

Семейством называется группа моделей компьютеров, которые используют одни и те же способы кодирования данных, имеют одинаковые системы команд центрального процессора и одинаковую логическую структуру. Отличаются модели друг от друга по скорости, объемам памяти, стоимости и т. д. В связи с этим разные модели могут использоваться для удовлетворения различных потребностей пользователей.

Поскольку машины, входящие в семейство, используют одни и те же способы кодирования данных, одну и ту же систему команд, можно осуществлять широкий обмен программами и данными между разными пользователями без внесения в программы существенных изменений.

В ходе развития семейства в архитектуру новых моделей могут вноситься усовершенствования, в систему команд могут включаться новые команды и группы команд. Но при этом практически всегда сохраняется так называемая *обратная совместимость*, означающая, что все созданные программы, работавшие на предшествующих моделях, без каких-либо изменений должны выполняться на любой новой модели.

Появление машин третьего поколения принято датировать 1964 г., когда были выпущены первые модели семейства IBM/360. Всего было выпущено одиннадцать различных по своим техническим характеристикам, но одинаковых по архитектуре моделей этого семейства. Популярность машин семейства IBM/360 оказалась настолько велика, что во всем мире их стали копировать или выпускать похожие по функциональным возможностям и совпадающие по способам кодирования

и обработки информации. Причем программы, подготовленные для выполнения на машинах IBM, с успехом выполнялись на их аналогах, так же как и программы, написанные для выполнения на аналогах, могли быть выполнены на машинах IBM. Такие модели машин принято называть *программно-совместимыми*. В нашей стране такой программно-совместимой с семейством IBM/360 была серия машин ЕС ЭВМ, в которую входило около двух десятков различных по мощности моделей.

Еще одним важнейшим нововведением этого периода было появление операционных систем и систем управления базами данных, без которых немыслимо представить сегодняшние информационные технологии.

Начиная с третьего поколения, вычислительные машины становятся повсеместно доступными и широко используются для решения самых различных задач. Характерным для этого времени является коллективное использование машин, так как они все еще довольно дороги, занимают большие площади и требуют сложного и дорогостоящего обслуживания. Правда, доступ к возможностям машины уже организуется и с индивидуально используемых устройств — **терминалов** (от terminal — конечный пункт), которые представляют собой оконечные устройства, состоящие из клавиатуры и дисплея и обычно не имеющие собственного процессора и памяти. Терминалы находятся на некотором удалении от основного оборудования машины, иногда даже на рабочих местах пользователей в других помещениях, и используются как устройства удаленного ввода и вывода для полноценного компьютера.

Основными носителями обрабатываемых данных все еще являются перфокарты и перфоленты, хотя уже значительный объем информации сосредоточивается на магнитных носителях — дисках и лентах.

В первой половине 1970-х гг. происходит переход от обычных интегральных схем к схемам с большей плотностью монтажа — большим интегральным схемам (см. 3.4). На фоне этого перехода произошло разделение до этой поры, в общем-то, единого потока развития средств вычислительной техники на две ветви.

Одна ветвь продолжала тенденцию наращивания мощности и надежности, а также коллективного использования вычислительных мощностей. Считается, что машины этого направления образуют четвертое поколение компьютеров. Среди них следует упомянуть семейство машин IBM/370, а также модель IBM 196, скорость которой достигла 15 млн. операций в секунду. Отечественными представителями машин четвертого поколения являются машины семейства «Эльбрус». Отличительная черта четвертого поколения — наличие в одной машине нескольких (обычно 2–6, иногда до нескольких сотен и даже тысяч) центральных процессоров, которые могут дублировать друг друга или независимым образом выполнять вычисления. Такая структура позволяет резко повысить надежность машин и скорость вычислений.

Вторая ветвь развития средств вычислительной техники оказалась направленной на миниатюризацию и персонализацию средств обработки данных. Своим рождением это направление обязано появлению в 1971 г. первого микропроцессора

Intel 4004. Для микропроцессоров введена отдельная классификация, по которой Intel 4004 относится к первому поколению микропроцессоров. Основные этапы развития микропроцессорной техники рассматриваются в главе 16.

Другая важная особенность этого периода связана с появлением мощных средств, обеспечивающих работу компьютерных сетей. Это позволило впоследствии создавать и развивать на их основе глобальные и локальные компьютерные сети.

Последним на сегодняшний день считается пятое поколение компьютеров. О проекте создания машин этого поколения, рассчитанном на десять лет, объявили в начале 1980-х гг. японские разработчики. За ними в эту стратегическую гонку втянулись ученые многих стран мира, в том числе США, СССР и ряда стран Западной Европы. Было заявлено, что к началу 90-х гг. будет создано принципиально иное по стилю обработки информации и взаимодействия с пользователем поколение машин. Если в традиционных информационных технологиях человек тщательно и подробно формулирует машине последовательность действий по обработке информации, то в проекте пятого поколения машина должна по поставленной перед ней цели самостоятельно составить план действий и выполнить его. Планировалось ввести общение с машиной на уровне естественного языка. Однако решить полностью весь комплекс задач проекта не удалось и до сих пор. Хотя имеются впечатляющие достижения по каждому из направлений проекта, возникли определенные технические трудности. Кроме того, усилия значительной части разработчиков были переключены на микропроцессорную технику и развитие сетевых технологий.

15.3. Функциональная классификация компьютеров

Классифицирующим признаком функциональной схемы является область применения вычислительных средств. Это наиболее размытая схема, потому что обычно существует несколько примерно равноценных вариантов выбора модели компьютера для решения поставленной задачи. Вместе с тем имеются области преимущественного использования моделей тех или иных классов. Они-то и составляют некую устойчивую основу для функциональной классификации. В настоящее время принято выделять следующие группы: суперкомпьютеры, универсальные компьютеры, мини-компьютеры, персональные компьютеры и встраиваемые процессоры.

Суперкомпьютер, или *суперЭВМ*, представляет собой сверхмощную одиночную обычно многопроцессорную вычислительную систему, которая способна решать задачи предельных классов, то есть такие задачи, в которых приходится с максимально возможными скоростями обрабатывать огромные массивы данных. К ним относятся, например, задачи метеопрогноза в планетарных масштабах, управления системами противоракетной и космической обороны, расчета и проектирования современных самолетов и космических кораблей, задачи из области ядерной

физики и космогонических исследований, задачи обеспечения работы глобальных сетей общемирового значения и т. д. Для их решения и достижения необходимого для этого уровня производительности суперкомпьютеры содержат десятки тысяч процессоров. Стоимость суперкомпьютеров может доходить до 500 млн. долларов и более.

По состоянию на лето 2005 г. мощнейшим суперкомпьютером в мире была состоящая из 65 536 процессоров американская вычислительная система Blue Gene/L, пиковая производительность которой в это время равнялась 136,8 Тфлоп (136,8 трлн. операций с плавающей точкой в секунду). Отметим, что проектная скорость этого суперкомпьютера 360 Тфлоп. Состоящая из 924 процессоров отечественная машина МВС-1500/ВМ с пиковой производительностью 8,1 Тфлоп в списке мощнейших машин мира в это же время занимала неплохое 56-е место. Заметим, что в это время во всем мире имелось лишь девять суперкомпьютеров, максимальное быстродействие которых превышало 10 Тфлоп. В настоящее время уже проектируются системы, скорость работы которых будет измеряться петафлопами — квадриллионами флопов.

Группа *универсальных компьютеров*, или **мэйнфреймов** (от mainframe — главный каркас, центральное строение), характеризуется возможностью решать подавляющее большинство задач обработки информации и практически неограниченными возможностями ее хранения. Универсальные машины представляют собой большие, обычно однопроцессорные или с относительно небольшим количеством центральных процессоров, вычислительные системы. Они используются как центральное звено в системах управления производственным циклом, в вычислительных центрах крупных предприятий, высших учебных заведений, исследовательских центров, а также как массовые хранилища информации. В последнее время универсальные компьютеры часто применяются в качестве ведущего элемента глобальных и локальных сетей, который предоставляет свои вычислительные ресурсы всем подключенным к сети компьютерам. К группе универсальных компьютеров относят машины типа ЕС ЭВМ и другие аналогичные им. Эта группа машин постепенно вытесняется мощными персональными компьютерами.

Мини-компьютеры, или **мини-ЭВМ**, использовались для работы в условиях реального производства для управления поточными линиями, как центральное звено в системах управления оборудованием цеха или отдела в больших организациях, а также для обеспечения работы средних по размерам организаций. Мини-компьютеры представляли собой более дешевый вариант универсальных машин. Как правило, мини-компьютеры выполнялись в виде нескольких напольных стоек, содержащих все устройства. В настоящее время эта группа машин считается устаревшей, они практически полностью вытеснены более мощными и дешевыми персональными компьютерами.

Персональные компьютеры (устаревшие названия — **микрокомпьютеры**, **микроЭВМ**) — это группа машин настольного исполнения, которые эксплуатируется, как правило, одним человеком или небольшим коллективом специалистов для решения своих профессиональных задач. Иногда персональный компьютер используется как ведущий элемент системы управления группой механизмов.

По своим вычислительным возможностям современные персональные компьютеры оставили далеко позади себя машины второго и третьего поколений, не говоря уже о машинах первого поколения. Для большей наглядности вы можете сравнить тридцатитонный «динозавр» ENIAC с его размерами и скоростью в 5 тыс. операций в секунду и стандартный современный персональный компьютер, умещающийся на рабочем столе специалиста и выполняющий миллиарды операций в секунду.

Встроенные процессоры представляют собой программируемые интегральные схемы, включаемые в конструкцию какого-либо отдельного устройства или механизма (автомобиль, металлорежущий станок, крылатая ракета) с целью автоматизации управления или оптимизации его работы. Если такая микросхема соединяется с какими-либо запоминающими устройствами и/или устройствами обмена данными, то такую конструкцию называют встроенным компьютером. Оказывается, что выгоднее встраивать в различные устройства и механизмы по-разному запрограммированные, но однотипные процессоры или компьютеры, чем для каждого из них заново разрабатывать уникальные устройства управления.

- Существует еще одна классификация, основанная на сетевой модели «клиент-сервер». Компьютеры, которые предоставляют свои ресурсы другим компьютерам, принято называть *серверами* (от serve — обслуживать, быть полезным), а машины, которые эти ресурсы используют, — **клиентами**. С этой точки зрения в группу серверов попадают подключенные к сети суперкомпьютеры, универсальные компьютеры и мощные персональные машины, которые в этом случае называются рабочими станциями. В группу клиентов входят персональные компьютеры, играющие роль так называемого *интеллектуального терминала* — более мощного, чем обычный терминал, устройства, которое не только обеспечивает обмен данными или управление работой компьютера в сети, но и может взять на себя значительную часть функций по хранению и обработке информации. Кроме того, к группе клиентов относятся так называемые *бездисковые* рабочие станции — персональные машины без внешней памяти, а также терминальные установки — оконечные устройства, состоящие из клавиатуры и дисплея и обычно не имеющие собственного процессора и памяти.
- К настоящему времени все более четко просматриваются глобальные изменения функциональной классификации компьютеров, в результате которых в этой схеме, возможно, останутся только два класса: серверы и клиенты. Внутри этих двух классов, скорее всего, появятся дополнительные градации, например: домашний сервер, сервер локальной сети, интернет-сервер и т. д.

15.4. Классификация персональных компьютеров

В 1999 г. был введен в действие международный стандарт «спецификации PC99», который определяет классификацию, а также требования к аппаратным и программным средствам персональных компьютеров.

ВНИМАНИЕ

Спецификацией называется формализованное (то есть выполненное по некоторой форме, с соблюдением определенных правил) описание свойств, характеристик и функций некоторого объекта.

Сразу же отметим, что классификация персональных компьютеров, предложенная в стандарте PC99, сохранилась и в стандартах, принятых в последующие годы (PC2000 — PC2004). Согласно указанным стандартам, вводится пять категорий персональных компьютеров (в скобках указаны соответствующие официальные термины).

- **Домашний** пользовательский, потребительский, массовый компьютер (**Consumer PC**), предназначен для работы в основном в домашних условиях и не имеет повышенных или пониженных требований к аппаратным и программным средствам.
- **Офисный**, деловой компьютер (**Office PC**) предназначен для выполнения канцелярской работы в составе компьютерных сетей предприятия, организации и т. д. Имеет минимальные требования к графике, работа со звуком вообще не предусматривается. Домашние и офисные компьютеры часто объединяют в одну группу настольных компьютеров.
- **Мобильный**, переносной, портативный компьютер (**Mobile PC**) предназначен для специалистов, которые используют информационные технологии в поездках, во время деловых встреч и т. д., когда доступ к стационарным машинам затруднен или вообще невозможен. Мобильные компьютеры обладают малым весом, небольшими размерами и автономным питанием. Обязательным считается наличие средств удаленного доступа. В эту группу входят **ноутбуки** (от notebook — записная книга), портативные компьютеры, или **лэптопы** (от laptop — «наколенный»), ручные или карманные компьютеры (КПК), или **палмтопы** (от palmtop, еще их называют hand-held PC или HPC — дословно: на поверхности ладони, наладонный). Приведем некоторые данные для сравнения этих групп компьютеров. Вес ноутбука равен примерно 2–3 кг, а вес КПК обычно составляет 100–300 г. Размеры ноутбука: длина 30–50 см при толщине 5–10 см; карманный компьютер имеет длину 10–15 см при толщине 1–2 см.

  Существуют также компьютеры, которые можно отнести к промежуточной группе между настольными и мобильными компьютерами. Эти компьютеры иногда называют *настольными мини-компьютерами* (Book PC — книжный персональный компьютер; slim-deck — тонкий настольный). По размерам они в 2–3 раза меньше обычного настольного компьютера.
- **Рабочая станция** (**Workstation PC**) используется в качестве сервера в компьютерных сетях, а также как инструмент разработчиками программных средств, конструкторами машин и механизмов, в книжных издательствах и т. д., то есть там, где предъявляются повышенные требования к ресурсам компьютера, к его вычислительной мощности и объемам хранимых данных.
- **Игровой**, или развлекательный, компьютер (**Entertainment PC**) используется для игр, а также для высококачественной работы со звуком и видеозаписями.

15.5. Классификация по архитектуре системы команд

Эта схема является, пожалуй, самой важной с точки зрения архитектуры компьютера, так как классифицирующими признаками являются структура и особенности системы команд процессора. На рис. 15.1 представлено современное состояние дерева архитектур по особенностям системы команд.

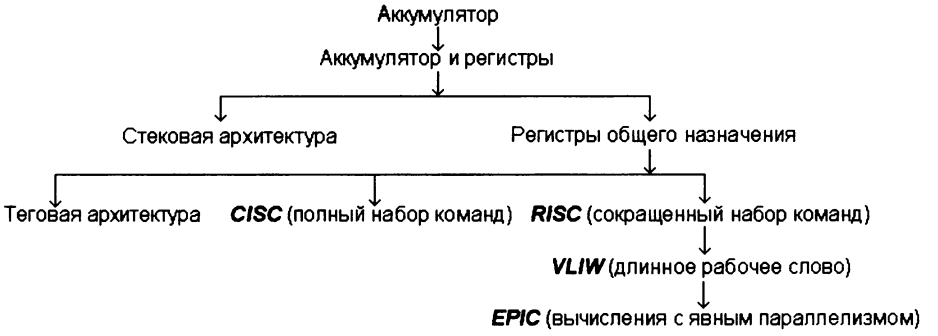


Рис. 15.1. Классификация компьютеров по архитектуре системы команд

Архитектура по фон-нейману является исходной точкой развития архитектуры всех современных цифровых машин. Поэтому можно считать, что предложенная им и обсуждавшаяся в 5.2 аккумуляторная архитектура образует корень дерева классификации архитектур.

15.5.1. Аккумуляторная архитектура

В этой архитектуре основу арифметико-логического устройства составляет единственный регистр, который называется аккумулятором. Система команд машины одноадресная. Типичная команда выбирает операнд из оперативной памяти (ОП) и использует его в операции с аккумулятором, где при необходимости уже должен находиться второй операнд. Результат остается в аккумуляторе.

Система имеет две команды, которые могут работать с оперативной памятью. Одна команда служит для загрузки кода в аккумулятор процессора из оперативной памяти, а вторая — для записи кода из аккумулятора в поле памяти.

Типичный пример последовательности машинных команд для такого процессора имеет вид

```
load Pa  ;загрузка в аккумулятор из ОП
add Pb   ;сложение
store Pc ;запись результата в ОП
```

Здесь `Pa` и `Pb` представляют адреса полей оперативной памяти, в которых находятся слагаемые, а `Pc` — адрес поля для записи результата.

Дальнейшее развитие привело к появлению архитектуры «аккумулятор и регистры», в которой в состав процессора наряду с аккумулятором входили несколько дополнительных регистров. Они использовались в основном как индексные. Впоследствии из этого типа архитектуры выделились две самостоятельные группы: *стековая* архитектура и архитектура *регистров общего назначения*.

15.5.2. Стековая архитектура

Команды стековой архитектуры в основном относятся к безадресным. Для хранения операндов используется входящий в состав процессора стек. Оба операнда выбираются из вершины стека, над ними выполняется задаваемое командой действие, результат остается в качестве новой вершины стека. В систему команд входят две одноадресные команды, которые обеспечивают копирование кода в вершину стека из поля оперативной памяти, а также выборку кода из вершины стека и запись в поле оперативной памяти.

Типичный пример последовательности машинных команд для такого процессора имеет вид

```
push Pa ;запись первого операнда в вершину стека из ОП
push Pb ;запись второго операнда в вершину стека из ОП
add     ;сложение
pop Pc  ;выборка результата из вершины и запись  в ОП
```

Стековая архитектура в чистом виде не нашла широкого применения. Ее отдельные, наиболее интересные и перспективные решения вошли в качестве составной части в другие, более поздние варианты архитектур.

15.5.3. Архитектура регистров общего назначения

Архитектура этого типа отличается тем, что процессор располагает набором высокоскоростных регистров, который принято называть **регистровой памятью** или *регистровым файлом*. В отличие от стековой архитектуры, операнды команд могут выбираться из любого регистра и записываться в любой регистр процессора. Функциональное, целевое использование регистров может быть произвольным. В главе 4 подробно рассмотрена архитектура системы на базе процессора i8086, который является типичным представителем обсуждаемой архитектуры.

Различают две большие группы архитектур регистров общего назначения: с *полным* и *усеченным* набором команд. Эти группы более подробно рассматриваются далее, однако сразу отметим, что оба варианта архитектур являются идеализированными, модельными представлениями. Исторически эти направления развивались как противоположные, но в настоящее время они практически неразличимы — любой реальный процессор фактически занимает некоторое промежуточное положение с преобладанием особенностей той или иной модели.

К архитектурам регистров общего назначения относится и так называемая *теговая* архитектура, отличительной особенностью которой является наличие связанного с каждым стандартным полем оперативной памяти аппаратно управляемого и анализируемого тега. В данном случае тег — это уже не номер страницы, как в организации кэша, а некоторый признак, определяющий типы данных, допустимых для характеризуемого тегом поля, а также множество применимых к этим данных машинных команд. Теговую архитектуру имеет система команд отечественной вычислительной системы «Эльбрус».

15.5.4. CISC-архитектура

До середины 1990-х гг. преобладающей считалась архитектура, которую принято обозначать аббревиатурой *CISC*, что расшифровывается как Complete Instruction Set Computer — компьютер с полным набором инструкций (машинных команд). Имеется и другой вариант расшифровки этого сокращения: Complete Instruction Set Code — полный набор кодов команд. Как следует из названия, отличительной особенностью этой разновидности архитектуры является наличие отдельной машинной команды для каждого возможного действия по обработке данных. Для архитектуры CISC характерно:

- относительно небольшое количество регистров общего назначения (например, всего четыре регистра в машинах с процессором i8086);
- большое количество различных машинных команд, каждая из которых выполняется за несколько тактов центрального процессора;
- различные форматы команд с разной длиной;
- преобладание двухадресной адресации;
- развитый механизм адресации операндов, включающий различные методы косвенной адресации.

К архитектуре CISC относятся системы команд семейств IBM 360/370, VAX, Intel 80x86, в том числе система команд подробно рассматривавшейся в главе 4 машины IBM PC с процессором i8086. В 16.1 более детально обсуждаются особенности семейства Intel в целом.

15.5.5. RISC-архитектура

Из-за наличия у CISC-архитектуры определенных недостатков, преодоление которых началось с появлением микроархитектурного уровня, к середине 1980-х гг. сформировалось новое архитектурное направление, которое назвали *RISC*-архитектурой (от Reduced Instruction Set Computer или Code — компьютер с сокращенным набором инструкций или усеченный набор кодов команд).

В этом подходе предполагается включение в систему команд процессора только часто встречающихся простых действий. Реализация более сложных операций над данными осуществляется с помощью последовательностей простых команд.

Фактически система команд такого компьютера содержит только микрокоманды (см. 6.5.1). Для архитектуры RISC характерно:

- большое количество регистров в составе процессора;
- упрощение и сокращение набора команд;
- одинаковый формат для преобладающего количества команд;
- одна и та же разрядность для всех команд;
- выполнение командами простых действий, как правило, за один такт;
- использование для обрабатывающих команд (сложения, умножения и т. д.) только регистровой адресации;
- использование только простых способов адресации (регистровая, прямая, непосредственная);
- минимизация времени выполнения команд;
- отделение команд обработки от команд обращения к памяти, то есть обращение к оперативной памяти только с помощью команды загрузки в регистр и команды сохранения содержимого регистра в оперативной памяти.

Следует отметить, что для компьютеров, относящихся к RISC-архитектуре, требуются более совершенные компиляторы, по сравнению с компьютерами CISC-архитектуры, которые предусматривают большое количество машинных команд, аналогичных операторам в высокоуровневых языках программирования.

К RISC-архитектуре относятся семейства SUN SPARC, Alpha, Power PC, MIPS и некоторые другие. В 16.2 более детально обсуждаются особенности RISC-машин семейства SUN SPARC.

15.5.5. VLIW-архитектура

Архитектура *VLIW* (от Very Large Instruction Word — очень длинное командное слово) является разновидностью RISC-архитектуры. Архитектура процессоров этой группы — суперскалярная, с наличием большого количества арифметико-логических функциональных блоков. Основным ее отличием является возможность объединения нескольких простых команд в так называемую *связку*. Входящие в нее команды должны быть независимы друг от друга, то есть их можно выполнять *одновременно, параллельно*. Таким образом, из нескольких независимых машинных команд транслятор формирует одно «очень длинное командное слово». Например, в системе команд отечественного суперкомпьютера «Эльбрус-3» командное слово занимает 256 бит в упакованном виде и 500 бит в распакованном представлении. Одно командное слово задает до семи арифметических или логических операций.

Подчеркнем, что анализ возможности объединения машинных команд в связку осуществляется на стадии трансляции. Поэтому во время выполнения программы процессору не приходится выполнять переупорядочение команд с целью исключения простоев конвейера. Однако процессоры архитектуры VLIW используют специальную систему команд, которая допускает проведение такого анализа

и потому несовместима с системой команд процессоров семейства Intel 80x86. Типичными представителями компьютеров VLIW-архитектуры являются семейства Multiflow и Cydra.

15.5.6. EPIC-архитектура

Архитектура *EPIC* (от Explicitly Parallel Instruction Computing — обработка команд с явным параллелизмом) является развитием VLIW-архитектуры. Она была разработана совместно компаниями Intel и Hewlett-Packard (HP). Отличительной чертой EPIC-архитектуры является устранение замеченных недостатков VLIW-архитектуры, требовавших, например, включения группы пустых команд для заполнения машинных тактов, возникающих при реализации параллельного исполнения некоторых команд. Характерные особенности EPIC-архитектуры:

- хорошая масштабируемость функциональных блоков процессора;
- явно задаваемый параллелизм в машинном коде;
- предикатное выполнение команд.

ВНИМАНИЕ

Под масштабируемостью в архитектуре компьютеров понимается возможность значительного увеличения количества каких-либо узлов, блоков, устройств с сохранением исходного режима функционирования компьютера без перенастройки оборудования и без необходимости изменения используемого программного обеспечения.

- Даже небольшое изменение количества подключаемых однотипных или разнотипных устройств порождает значительные конструктивные трудности, связанные с необходимостью иметь хотя бы достаточное количество разъемов для их подсоединения. Имеются также проблемы с обеспечением всех устройств электропитанием, с распределением информационных потоков для их безостановочной деятельности, с выбором приоритетного устройства при одновременном запросе одного и того же ресурса, а также множество других проблем, связанных со значительным наращиванием количества устройств. Вместе с тем количество процессоров, входящих в состав одной вычислительной системы, уже в настоящее время исчисляется десятками тысяч. Поэтому проблеме масштабируемости в архитектуре EPIC уделяется много внимания.
- Явное задание параллелизма в команде EPIC-архитектуры, как и в архитектуре VLIW, призвано упростить и, соответственно, ускорить работу процессора, а предикатное выполнение фактически превращает обработку ветвлений в параллельное выполнение нескольких линейных фрагментов программы, что избавляет конвейерную схему от проблем с очисткой и заполнением конвейера. Примеры машин EPIC-архитектуры более детально обсуждаются в 16.1.

15.6. Прочие классификационные схемы

Кратко обсудим другие классификационные схемы. Классификация по разрядности машинного слова выделяет в одну и ту же группу машины любых архитектур, но с одной и той же длиной машинного слова. Как уже упоминалось, название такой архитектуры образуется от длины слова в битах. Так, исторически первыми персональными машинами были 8-битовые компьютеры, им на смену пришли рассмотренные в главе 4 более мощные 16-битовые. В настоящее время преобладают 32-битовые компьютеры, обзору архитектурных особенностей которых посвящен раздел 4.2.8. Однако они уже начинают сдавать позиции 64-битовым машинам, особенности архитектуры которых кратко рассматриваются в 16.1.7.

В классификации по типу управления выделяются две группы архитектур. В одной группе, к которой относятся все **классические** (традиционные) архитектуры, последовательность действий процессора задается потоком машинных команд, образующих выполняемую им программу. Вторую группу образуют архитектуры, которые в настоящее время относятся к **неклассическим** (нетрадиционным). В этих вычислительных системах последовательность действий определяется потоком обрабатываемых данных. Особенности неклассических архитектур рассматриваются в главе 18.

Классификации по степени параллелизма посвящена глава 17, в которой обсуждаются используемые при этом классифицирующие признаки и выделяемые группы вычислительных систем.

Дополнительный материал по обсуждавшимся в главе вопросам можно найти в изданиях [2], [5], [20], [22].

Контрольные вопросы и упражнения

1. Перечислите и охарактеризуйте основные классификационные схемы вычислительных систем.
2. Дайте сравнительную характеристику аналоговых и цифровых вычислительных систем.
3. Укажите классификационные признаки в классификации вычислительных систем по поколениям.
4. Дайте сравнительную характеристику первого и второго поколений вычислительных систем.
5. Укажите основные отличия третьего поколения вычислительных систем.
6. Поясните смысл терминов «семейство», «обратная совместимость», «программная совместимость».
7. Охарактеризуйте четвертое и пятое поколения вычислительных систем.

8. Дайте сравнительную характеристику классов суперкомпьютеров и универсальных машин.
9. Какие классы выделяются в международной классификации персональных компьютеров? Приведите их сравнительную характеристику.
10. Изобразите дерево классификации архитектур по системе команд.
11. Охарактеризуйте аккумуляторную, стековую и теговую архитектуры.
12. Дайте характеристику CISC-архитектуры. Приведите примеры систем, относящихся к ней.
13. Дайте характеристику RISC-архитектуры. Приведите примеры систем, относящихся к ней.
14. Дайте сравнительную характеристику CISC- и RISC-архитектур. Какая из них лучше и имеет больше перспектив развития?
15. Чем отличается VLIW-архитектура от RISC-архитектуры?
16. Сравните между собой VLIW- и EPIC-архитектуры.
17. Что понимается под масштабируемостью вычислительных систем?
18. Какие классы машин выделяются в классификации по типу управления?

Глава 16
Обзор основных семейств микропроцессоров

В этой главе обсуждаются технические и архитектурные особенности наиболее популярных в настоящее время семейств микропроцессоров, которые используются в персональных компьютерах и в вычислительных системах других классов.

16.1. Семейство Intel

Компьютеры IBM PC, реализованные на относящихся к CISC-архитектуре процессорах семейства Intel, в настоящее время являются наиболее широко распространенными в мире. В последовательно выпускаемых моделях этого семейства можно отчетливо проследить основные этапы развития архитектуры вычислительных систем. Некоторые технические характеристики моделей, входящих в семейство, приведены в табл. 16.1.

16.1.1. Первые модели процессоров Intel

История семейства началась с выпуска компанией Intel в 1971 г. 4-битового процессора i4004. Он состоял из 2300 транзисторов и изготавливался по 10-микронной технологии, то есть линейный размер вентиля был приблизительно равен десяти микрометрам. Стоил процессор всего 200 долларов и помещался на ладони человека, в то время как его вычислительная мощность была сопоставима с мощностью созданного в 1946 г. компьютера ENIAC, который размещался в помещении площадью 120 м^2 и стоил несколько миллионов долларов. Тактовая частота процессора составляла 108 КГц, быстродействие равнялось 0,06 MIPS (60 000 арифметических операций в секунду), а объем адресного пространства был равен всего 640 байт.

Таблица 16.1. Основные модели процессоров семейства Intel

Модель	Год	Количество транзисторов	Тактовая частота	Объем адресного пространства
i4004	1971	2300	108 кГц	640 байт
i8008	1972	3500	200 кГц	16 Кбайт
i8080	1974	6000	2 МГц	64 Кбайт
i8086	1978	29 000	4,77–10 МГц	1 Мбайт
i8088	1979	29 000	4,77–8 МГц	1 Мбайт
i80286	1982	134 000	6–12 МГц	16 Мбайт
i80386	1985	275 000	16–33 МГц	4 Гбайт
i80486	1989	1 200 000	25–66 МГц	4 Гбайт
Pentium (P5)	1993	3 100 000	60–200 МГц	4 Гбайт
Pentium Pro (P6)	1995	5 500 000	150–200 МГц	64 Гбайт
Pentium MMX (P6)	1996	5 500 000	150–200 МГц	64 Гбайт
Pentium II (P6)	1997	7 500 000	233–450 МГц	64 Гбайт
Pentium III (P6)	1999	20 000 000	До 1000 МГц	64 Гбайт
Pentium IV (P6)	2000	42 000 000	1,3–2,0 ГГц	64 Гбайт
Pentium Extreme Edition	2004	125 000 000	До 3,4 ГГЦ	2 Тбайт

Через год был выпущен 8-битовый процессор i8008, состоявший из 3500 транзисторов (10-микронная технология), работавший на частоте 0,2 МГц с быстродействием 0,06 MIPS и обеспечивавший работу с 16 Кбайт оперативной памяти.

В 1974 г. фирма Intel выпустила 8-битовый процессор i8080, состоявший из 6000 транзисторов (6-микронная технология). Он работал на частоте 2 МГц, а его быстродействие составляло 0,64 MIPS, то есть более чем в 10 раз выше, чем у i8008. Процессор имел 8-битовую шину данных и 16-битовую шину адреса, следовательно, объем адресного пространства составлял 64 Кбайт.

Этот процессор оказал значительное воздействие на развитие вычислительной техники во всем мире. В частности, на базе i8080 был создан персональный компьютер Altair 8800, который завоевал мировую известность. Он продавался за 400 долларов как комплект устройств, состоящий из процессора, оперативной памяти объемом 256 байт, шинного устройства и блока питания, из которых покупатель должен был сам собрать компьютер. Для него Биллом Гейтсом и Полом Айленом, которые позднее основали компанию Microsoft, был создан первый интерпретатор с языка Бейсик.

16.1.2. Шестнадцатибитовые модели семейства Intel

В 1978 г. был выпущен 16-битовый процессор i8086, ставший базовым для всего семейства процессоров Intel. Он состоял из 29 000 транзисторов (3-микронная технология), работал на частоте 4,77 МГц, и его производительность была равна

0,33 MIPS. Позднее частота увеличилась до 10 МГц, и до 0,75 MIPS возросла производительность. Предусматривалась работа с 16-битовой шиной данных и 20-битовой адресной шиной, обеспечивавшей адресацию одного мегабайта сегментированной оперативной памяти. Архитектура процессора i8086 рассматривалась в главе 4.

К моменту выпуска 16-битового процессора i8086 уже довольно много вычислительных устройств работало на 8-битовом процессоре i8080, имевшем 8-разрядную шину данных. Чтобы обеспечить возможность перехода в этих устройствах к 16-битовой архитектуре, компания Intel в 1979 г. выпустила процессор i8088, по архитектуре и техническим характеристикам полностью аналогичный i8086, но имевший 8-разрядную мультиплексированную шину данных, по которой машинное слово передавалось за два такта — сначала один байт, затем второй. А для потребителей, которых интересовали вычисления с плавающей точкой, в 1980 г. был выпущен **математический сопроцессор** i8087, обеспечивавший с помощью специализированной системы команд выполнение операций над данными в формате с плавающей точкой.

На основе процессора i8088 фирмой IBM в 1981 году был создан персональный компьютер International Business Machines 5150 Personal Computer (IBM PC). Он имел 40 Кбайт оперативной памяти с возможностью расширения до 640 Кбайт, четырехцветный дисплей и накопитель на гибких дисках объемом 160 Кбайт.

По соглашению с фирмой IBM для этого компьютера компания Microsoft выпустила операционную систему, которая впоследствии получила название MS DOS (от Microsoft Disk Operation System).

Популярность компьютеров IBM PC оказалась огромной — за первые восемь месяцев компания продала 50 000 экземпляров. Все спецификации этого компьютера были открытыми (опубликованы), это дало толчок производству другими компаниями IBM PC-совместимых машин.

В 1983 г. на базе процессора i8088 был начат выпуск компьютеров модели IBM PC/XT (от eXtended Technology — расширенная технология) с объемом оперативной памяти 128 Кбайт и выше. В состав машины входили дисковод гибких дисков объемом 360 Кбайт и жесткий диск объемом 10 Мбайт. В качестве дополнительной возможности предусматривалось подключение математического сопроцессора i8087.

Процессоры i8086, i8087 и i8088 относят к первому поколению микропроцессоров. К нему же следует отнести и не нашедшие широкого применения микропроцессоры моделей i80186 и i80188, в которых базовая архитектура i8086/i8088 была расширена включением в систему нескольких новых команд и появлением контроллера DMA.

В 1982 г. компания Intel начала выпуск микропроцессора второго поколения — 16-битового i80286. Он состоял из 134 000 транзисторов (1,5-микронная технология), работал на частотах до 12 МГц и имел производительность до 2,6 MIPS. Предусматривалась работа с 16-битовой шиной данных и 24-битовой адресной шиной, обеспечивающей объем адресного пространства 16 Мбайт.

Основу системы команд процессора i80286 составляла полностью вошедшая в нее система команд базового процессора i8086. Поэтому на компьютере с процессором i80286 можно выполнить любые программы, разработанные для базовой модели i8086. С тех пор компания Intel проводит политику программной совместимости любой старшей модели (то есть выпущенной позднее) со всеми младшими моделями (то есть выпущенными ранее). Так сформировалось семейство программно совместимых моделей процессоров (i8086, i80286, i80386, i80486, Pentium...), которое известно под названием **семейство i80x86**, или просто **x86**, где буква **x** заменяет одну из цифр маркировки процессора.

Важным нововведением в процессоре i80286 было появление многопрограммного защищенного режима работы. Для его обеспечения в систему команд процессора включили несколько новых групп команд. Кроме того, в нее вошли команды синхронизации, обеспечивающие взаимодействие с сопроцессором, а также команды взаимодействия с операционной системой и команды поддержки языков высокого уровня.

На базе процессора i80286 фирма IBM в 1984 г. выпустила персональный компьютер IBM PC/AT (от Advanced Technology — передовая технология) с объемом оперативной памяти 256 Кбайт, двумя дисководами гибких дисков объемом до 1,2 Мбайт, жестким диском объемом до 40 Мбайт и шестнадцатицветным дисплеем. Предусматривалась возможность доукомплектования компьютера выпущенным немного позднее математическим сопроцессором i20287.

16.1.3. Тридцатидвухбитовые модели i80386 и i80486

В 1985 г. компания Intel выпустила относящийся к третьему поколению 32-битовый микропроцессор i80386. Он состоял из 275 000 транзисторов (1,5-микронная технология), работал на частотах до 33 МГц с производительностью до 8,5 MIPS. Регистры центрального процессора, разрядность шины данных и адресной шины стали 32-битовыми, вследствие чего объем адресного пространства возрос до 4 Гбайт. Позднее для этой модели был разработан математический сопроцессор i80387. Унаследованная от процессора i80286 система команд в связи с переходом на 32-битовую модель была расширена командами, обеспечивающими операции с четырехбайтными данными. Появилось несколько новых форматов данных, форматов команд и способов адресации, максимальная длина команды увеличилась до 15 байт. Два новых префикса команд обеспечивали переключение между 16- и 32-битовыми длинами адресов и операндов. В архитектуре i80386 была реализована страничная организация виртуальной памяти объемом до 4 Тбайт и одноуровневая кэш-память для хранения наиболее часто используемых страниц.

В 1989 г. фирма Intel начала выпуск относящегося к четвертому поколению также 32-битового микропроцессора i80486. Он содержал 1 200 000 транзисторов и был выполнен по 1-микронной технологии. Впоследствии процессоры i80486 производились по 0,8- и 0,6-микронным технологиям. Процессор работал на частотах до 66 МГц и имел производительность до 54 MIPS. Модели процессоров i80386

и i80486 положили начало семейству процессоров, известному в настоящее время как **IA32** (от Intel Architecture 32 — 32-битовая архитектура Intel).

Анализируя приведенные данные, легко увидеть, что основным направлением развития процессоров от модели i4004 вплоть до модели i80386 было увеличение производительности компьютера за счет экстенсивного наращивания разрядности данных, тактовой частоты и объема оперативной памяти. В самом деле, за указанный период развития архитектуры разрядность увеличилась с 4 до 32 бит, тактовая частота выросла с 108 кГц до 33 МГц, а объем оперативной памяти увеличился с 640 байт до 4 Гбайт. Переломным моментом в развитии стала разработка процессора i80486, в архитектуру которого было включено несколько новых решений, связанных с параллелизмом. Эти новые подходы в архитектуре позволили существенно нарастить производительность процессора. Так, например, процессор i80386 с тактовой частотой 33 МГц обладал производительностью 8,5 MIPS, в то время как у процессора i80486 с той же тактовой частотой производительность была равна уже 27 MIPS.

Принципиально новыми моментами в архитектуре процессора i80486 стали:

- реализация процессора как пятиступенчатого конвейера;
- включение в состав системы двухуровневого кэша — внутреннего L1 объемом 8 Кбайт и внешнего L2 объемом 512 Кбайт.

Кроме того, начиная с процессора i80486, математический сопроцессор включается в общий корпус с центральным процессором и становится его составной частью. Для поддержки нового блока FPU (от Float Point Unit — блок плавающей точки — бывший сопроцессор) в состав процессора i80486 включили восемь регистров для работы с плавающей точкой объемом 80 битов каждый. Начиная с этой модели, во всех последующих процессорах семейства Intel предусматривается также возможность создания многопроцессорных вычислительных систем.

16.1.4. Пятое поколение моделей семейства

В начале 90-х гг. фирма Intel, сохраняя традиционную линию на программную совместимость с предыдущими моделями, начала разработку микропроцессоров пятого поколения. Первым результатом этого проекта был выпуск в 1993 г. процессора Pentium. Для этой и последующих моделей фирме Intel по юридическим мотивам пришлось отказаться от маркировки в стиле i80x86 и перейти к объявлению их словесных названий. Кроме того, в обиход вошло обозначение поколений процессоров вида P*n*, где *n* — номер поколения. В частности, процессор Pentium относят к пятому поколению, или к семейству P5.

Модели процессора Pentium изготавливались по 0,8-, 0,5- и 0,35-микронным технологиям, они состояли из 3 100 000 транзисторов, работали на частотах до 200 МГц и имели производительность до 330 MIPS. К этому времени приобрела популярность оценка мощности компьютера в тестовых единицах SPEC, в которых 200-мегагерцовый процессор Pentium имел показатели 5,17 SPECint95 и 4,32 SPECfp95.

Систему команд процессор Pentium унаследовал от процессора i80386. Внутренний кэш состоял из двух независимых банков — кэша данных и кэша команд — по 8 Кбайт каждый. Процессор содержал два конвейера и внутреннюю шину данных разрядностью 128 и 256 байт. При этом внешняя шина данных имела разрядность 64 бита, что обеспечивало ее повышенную пропускную способность. Другими нововведениями процессоров Pentium были введение суперскалярной архитектуры и предсказания адреса перехода.

Позднее фирма Intel приступила к производству процессоров Pentium MMX (от MultiMedia Technology — мультимедийные технологии), которые изготовлялись по 0,28-микронной технологии, состояли из 4,5 млн. транзисторов, работали на частотах до 233 МГц и имели на последней частоте показатели производительности 7,12 SPECint95 и 5,32 SPECfp95.

Процессоры Pentium MMX отличались от процессоров Pentium включенной в систему команд группой из 57 команд, обеспечивающих работу с мультимедийными данными, то есть с графикой и звуком. Для этих команд в состав процессора добавили восемь целочисленных регистров MMX, которые фактически были совмещены с регистрами сопроцессора. Объем внутреннего кэша увеличился до 32 Кбайт.

Считается, что использованная в Pentium MMX технология работы с мультимедийными данными является важным достижением в области архитектуры микропроцессоров. Эта технология улучшает сжатие и восстановление (компрессию и декомпрессию) изображения и звука, в том числе видеоданных (движущегося изображения в комбинации со звуком), обеспечивает шифрование/дешифрование, обработку ввода/вывода и т. д. В основе мультимедийного расширения системы команд процессора лежит технология обработки множественного потока данных в одной машинной команде, которую принято называть **SIMD** (от Single Instruction Multiple Data — одна инструкция, много данных). Современные мультимедийные и сетевые программы часто используют относительно короткие кратные (вложенные) циклы, выполнение которых отнимает до 90 % процессорного времени. В основанном на технологии SIMD подходе с помощью одной машинной команды одна и та же операция выполняется над группой данных. Это позволяет организовать вычисления с существенно меньшим количеством внутренних циклов и за счет этого увеличить производительность при выполнении соответствующих программ.

16.1.5. Шестое поколение моделей семейства

В 1995 г. компания Intel начала продажу микропроцессоров Pentium Pro (от professional — профессиональный), которые считаются первыми представителями шестого поколения (или семейства P6) микропроцессоров. Микропроцессор состоял из 5,5 млн. транзисторов, и его отличительной особенностью, характерной также для всех последующих моделей процессоров, стало объединение в одном корпусе с процессором кэш-памяти второго уровня объемом от 256 Кбайт до 1 Мбайт. Поэтому в зависимости от объема кэша процессор выпускался как по 0,5-, так и по 0,35-микронной технологии. Процессор работал на тактовых

частотах до 200 МГц, и имел показатели производительности 8,58 SPECint95 и 6,48 SPECfp95 на частоте 200 МГц с кэшем L2 объемом 512 Кбайт.

В процессор Pentium Pro было включено восемь 128-битовых регистров для MMX-данных в формате с плавающей точкой, он содержал три конвейера и поддерживал работу с 36-разрядной адресной шиной. Таким образом, объем адресного пространства возрос до 64 Гбайт. Однако главное отличие этого процессора состояло в том, что, оставаясь в целом в рамках CISC-архитектуры, компания Intel применила принципы *микроархитектуры*, похожие на принципы, использованные в RISC-архитектуре. Сложные команды семейства 80x86 в блоке микропрограммного управления процессора преобразовывались в последовательности простых микрокоманд. Таким образом был сделан первый шаг в объединении достоинств CISC- и RISC-архитектур в одном процессоре. Это усовершенствование позволило снять целый ряд ограничений, свойственных традиционной системе команд i80x86. Некоторые специалисты ввели для такого симбиоза архитектур обозначение x86-to-RISC86.

В 1997 г. началось производство процессора Pentium II, который объединил возможности процессоров Pentium Pro и Pentium MMX. На самом деле Pentium II — это обобщенное название целого семейства процессоров, которые предназначались для различных пользователей. В это семейство входили такие процессоры, как Klamath, Deschutes, Katmai и т. д., предназначенные для массового потребителя среднего уровня. Во вторую группу, известную под обобщающим названием Celeron, входили процессоры Covington, Mendocino, Dixon и др., обеспечивающие работу недорогих компьютеров. А в группу Xeon (Xeon, Tanner, Cascades и др.) входили микропроцессоры для высокопроизводительных серверов и рабочих станций.

Процессоры Pentium II изготавливались по 0,25- и 0,18-микронным технологиям и состояли из 7,5 млн. транзисторов. При тактовой частоте 450 МГц они имели показатели производительности 17,4 SPECint95 и 13,0 SPECfp95. Поддерживалась работа с внешним кэшем объемом от 512 Кбайт до 2 Мбайт, объем внутреннего разделенного кэша был равен 32 Кбайт.

Основными архитектурными особенностями процессора Pentium II стали двойная независимая шина DIP и технология динамического исполнения команд. Команды работы с мультимедийными данными, использующиеся в процессоре Pentium II, совместимы с аналогичными командами процессора Pentium. Однако их исполнение опирается на улучшенную архитектуру процессора и шины Pentium II. В работе кэша второго уровня L2 предусмотрено использование механизма коррекции ошибок ECC, который увеличивает надежность и целостность данных.

В 1999 г. фирма Intel выпустила процессор Pentium III, изготавливавшийся по 0,25-, 0,18- и 0,13-микронным технологиям и состоявший из 20 млн. транзисторов. Процессор работал на частотах от 500 МГц до 1 ГГц и имел на частоте 1 ГГц показатели производительности 442 SPECint2000 и 335 SPECfp2000, что соответствует 46,8 SPECint95 и 32,2 SPECfp95.

Основным архитектурным отличием Pentium III стало расширение системы команд группой **SSE** (от Streaming SIMD Extensions — потоковое расширение SIMD), состоящей из 70 новых команд, которые, в частности, включают SIMD-операции над четверками вещественных чисел одинарной точности и несколько новых целочисленных SIMD-операций. Эти команды обеспечивают возможность выполнения одной и той же операции над группой данных, что очень полезно, например, для организации перехода между различными системами координат во время обработки трехмерной графики. При использовании команд SSE вычисления, для которых до этого требовалось от четырех до шести отдельных машинных команд, можно задавать с помощью только одной команды. Все операции над мультимедийными данными были оптимизированы. Для поддержки команд над мультимедийными данными с плавающей точкой в процессор Pentium III добавили восемь 128-разрядных регистров.

В конце 2000 г. фирма Intel начала выпуск процессора Pentium 4, который изготавливался по 0,18-микронной технологии, содержал 42 млн. транзисторов, работал на частоте 1,4 ГГц, имел кэш L2 объемом 256 Кбайт и показатели производительности 535 SPECint2000 и 558 SPECfp2000.

Процессор Pentium 4 относится к суперконвейерной суперскалярной архитектуре с динамическим исполнением команд. Его конвейеры имеют глубину 20 ступеней. Кэш команд объемом 16 Кбайт содержит уже декодированную смесь микрокоманд, расположенных в переопределенном, готовом к выполнению порядке.

В состав системы команд включена группа **SSE2** из 144 новых команд для работы с мультимедийными данными. Эти команды включают операции над 128-битными целочисленными и 128-битными вещественными данными.

В 2002 г. был начат выпуск процессора Xeon класса Pentium 4 по 0,13-микронной технологии с поддержкой технологии Hyper Threading с тактовыми частотами до 2,2 ГГц и кэшем L2 объемом 512 Кбайт.

В 2004 г. фирма Intel представила свой первый микропроцессор, выполненный по 0,09-микронной (или 90-нанометровой) технологии Pentium Extreme Edition с тактовой частотой 3,4 ГГц. Он имеет трехуровневый кэш с объемами 32 Кбайт, 1 Мбайт и 2 Мбайт соответственно и работает с шиной памяти с тактовой частотой 800 МГц. Система команд дополнена группой **SSE3**, состоящей из 13 новых команд, которые обеспечивают ускоренное кодирование видео. Улучшена также поддержка технологии Hyper Treading.

В конце 2004 г. фирма Intel официально объявила о введении поддержки процессорами Pentium Extreme Edition и Xeon 64-разрядной адресации. Новая технология получила название **EM64T** (от Extended Memory 64-bit Technology — расширенная технология работы с 64-битовой памятью). Она позволила увеличить объем адресного пространства до 2 Тбайт. Кроме того, предусмотрены переход на новый стандарт шины PCI Express и работа с микросхемами памяти типа DDR2.

Хотя процессоры Pentium 4 в настоящее время все еще производятся и продаются, с большой долей уверенности можно утверждать, что эта модель является одной из последних в семействе процессоров шестого поколения. По-видимому,

возможности дальнейшего повышения эффективности архитектуры IA32 к началу XXI в. уже исчерпаны. Как считают специалисты, дальнейшее развитие связано с переходом к принципиально иной, 64-битовой архитектуре либо с созданием многоядерных процессоров. Компания Intel решила двигаться в обоих направлениях: уже в 1990-х гг. она приступила к разработке и двухъядерных систем, и 64-битовых процессоров, относящихся к седьмому поколению (к семейству P7).

16.1.6. Двухъядерные модели семейства Intel

Как оказалось впоследствии, выпуск Pentium 4 по технологии HP был шагом в направлении создания многоядерных процессоров, первый выпуск которых состоялся в апреле 2005 г. Были представлены состоящие из 230 000 000 транзисторов двухъядерные процессор Pentium D (от Desktop — настольный) и группа микропроцессоров Pentium EE 8xx в составе Pentium EE 820, Pentium EE 830 и Pentium EE 840. Новые процессоры построены по одной и той же архитектуре: два процессорных ядра, которые полностью аналогичны ядру, используемому в процессоре Pentium Extreme Edition (Pentium EE), находятся на одном кристалле и соединены с помощью внутренней шины. Каждое процессорное ядро имеет кэш L2 объемом 1 Мбайт. Тактовая частота ядер равна 2,8–3,2 ГГц.

По индивидуальной производительности каждое ядро в составе двухъядерного процессора Intel эквивалентно процессору Pentium EE с такой же, как у ядра, тактовой частотой. Однако их совместная работа, по-видимому, не приведет к удвоению производительности, поскольку обеспечить полную одновременную загрузку каждого процессора, скорее всего, невозможно. Существуют препятствующие этому ограничения как аппаратного, так и программного характера. Во-первых, должна быть обеспечена рассчитанная для двух ядер пропускная способность шин, а во-вторых, операционную систему, инструментальные среды и приложения необходимо перестраивать с учетом новых возможностей двухъядерных систем, связанных с параллельной работой ядер.

16.1.7. Особенности архитектуры IA64

Несмотря на появление в процессоре Pentium Pro элементов, характерных для RISC-архитектуры, полностью избавиться от недостатков, присущих 32-битовой CISC-архитектуре семейства Intel, не удалось. Вкратце их можно свести к следующему перечню проблем.

- В систему команд архитектуры IA32 входит большое количество форматов команд, которые облегчают трансляцию и обеспечивают возможность использования в программах различных типов данных. Вместе с тем различная длина команд разных форматов создает трудности в декодировании и исполнении, что приводит к замедлению выполнения программ.
- Система команд архитектуры IA32 содержит в основном двухадресные команды, определяющие до двух обращений в оперативную память каждая — одну для выборки операнда, а другую для записи результата. В настоящее время

более эффективными считаются системы команд с трехадресной регистровой адресацией, которые вообще не требуют обращения к оперативной памяти.

- Использованное в старших моделях процессоров Intel разбиение команд архитектуры CISC на микрокоманды требует дополнительных аппаратных средств и дополнительных временны́х затрат, усложняет разработку эффективных трансляторов.
- Архитектура IA32 в целом сохраняет унаследованную от i8086 модель, в которой в центральный процессор входит всего 4 регистра общего назначения. В принципе, к этой группе можно отнести еще два регистра, `esi` и `edi`, которые в 32-битовых машинах можно использовать произвольным образом (за исключением доступа к отдельным байтам). Тем не менее общее количество регистров общего назначения процессора в архитектуре IA32 очень мало, да к тому же они имеют различную внутреннюю структуру. Это вынуждает программистов при написании программ предусматривать дополнительные обращения к оперативной памяти, которые требуют дополнительных временны́х затрат.
- Из-за недостатка регистров в центральном процессоре существенно усложняется подготовка к прохождению взаимозависимых команд через конвейер процессора.
- Для обеспечения высокой скорости работы нужен конвейер процессора с большим количеством ступеней. Чем длиннее конвейер, тем сложнее предсказание переходов и тем выше дополнительные издержки, связанные с загрузкой и очисткой конвейера. Поэтому даже низкий процент неверных предсказаний существенно снижает общую производительность системы.
- Архитектура IA32 ограничивает размер программы адресным пространством объемом 64 Гбайт. Программы большего размера используют аппарат виртуальной памяти, что снижает производительность.

Эти недостатки стали проявляться уже в начале 1990-х гг. Однако избавится от большинства из них мешал принцип программной совместимости, неукоснительно соблюдавшийся при проектировании любой но́вой модели процессоров. Постоянно нараставший объем используемого программного обеспечения, стоимость которого превысила сотни миллиардов долларов, также способствовал сохранению традиционных подходов. Предполагалось, что огромное количество пользователей по всему миру с большой неохотой откажутся от привычных программ и вложат дополнительные средства в приобретение новых. Эти соображения довольно долго тормозили решительный отказ от подходов архитектуры IA32.

Впрочем, попытки повысить эффективность машин архитектуры IA32 наталкивались на все большие трудности. Проанализировав сложившуюся ситуацию, компания Intel пришла к выводу о том, что нужно построить абсолютно новую систему, основанную на новых представлениях и архитектурных решениях. Основные отличительные особенности новой архитектуры класса EPIC, которая получила название IA64, — следующие:

- система команд процессора состоит из трехадресных команд фиксированного формата с регистровой адресацией;

- стандартная команда занимает 40 битов и содержит код операции, два 6-битовых поля для указания регистров с операндами и одно 6-битовое поле для указания регистра результата;
- обращение к полям оперативной памяти осуществляется только с помощью отдельных специализированных команд;
- команда поступает в процессор в виде пучка из трех команд длиной 128 битов. Команды, входящие в пучок, могут выполняться параллельно различными функциональными блоками процессора;
- в процессоре предусмотрено большое количество однотипных функциональных блоков для организации параллельного выполнения команд;
- в состав центрального процессора входит 128 регистров общего назначения длиной 64 бита, 128 регистров с плавающей точкой длиной 82 бита, 64 предикатных однобитовых регистра, а также 128 специализированных 64-битовых регистров;
- имеется трехуровневый кэш с объемом L3 до 6 Мбайт.

Пучком команд в архитектуре IA64 называют группу команд, которые могут быть выполнены аппаратурой процессора одновременно. В простейшем случае пучок состоит из трех команд и 8-битового шаблона (рис. 16.1), содержащего информацию о том, какие команды могут выполняться параллельно. В принципе, несколько пучков с помощью входящего в шаблон бита могут образовать *связку*. Все команды, входящие в связку, также могут быть исполнены одновременно.

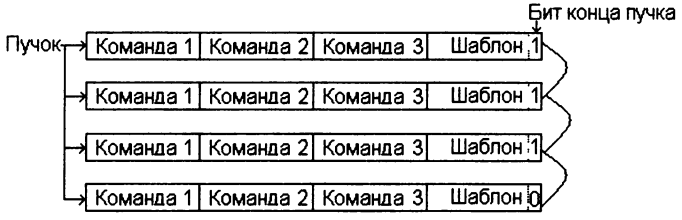


Рис. 16.1. Структура связки команд в архитектуре IA64

Можно отметить, что пучок команд, обеспечивающий явное задание параллелизма в программе (отсюда и название EPIC), является аналогом длинного командного слова архитектуры VLIW. В обеих архитектурах параллелизм задается явно в командах программы, то есть он выявляется еще на стадии трансляции, а не обнаруживается процессором в ходе выполнения программы, как это реализовано в процессорах CISC-архитектуры. В связи с этим напомним, что архитектура EPIC, к которой относится семейство IA64, является развитием VLIW- и RISC-архитектур.

Пучки формируются компилятором, который во время трансляции должен выделять команды, способные образовать пучок. Компилятор должен их переупорядочивать так, чтобы входящие в пучок команды не имели взаимозависимостей, не конфликтовали друг с другом за ресурсы и за операнды. Другими словами,

функция, которая выполнялась в архитектуре IA32 аппаратно, в EPIC-архитектуре перенесена на программный уровень. Это обеспечивает возможность упрощения аппаратуры, отвечающей за организацию работы конвейера.

Отметим, что трансляция выполняется только один раз, в то время как выполнять оттранслированную программу приходится многократно. Следовательно, целесообразно потратить больше времени на выявление параллелизма и оптимизацию программы на стадии трансляции. Исключение анализа команд на стадии выполнения и использование оптимизированного кода с явно заданными параллельными командами обеспечивает существенное ускорение выполнения программы.

Еще одной важнейшей, принципиально новой особенностью архитектуры IA64 по сравнению с традиционными RISC- и CISC-архитектурами является организация **предикатного** выполнения ветвлений, которое в любом случае избавляет от необходимости останавливать конвейер (точнее, разгружать и загружать его). Предикатное выполнение ветвления осуществляется так же, как и выполнение линейных участков программы.

В архитектуре IA64 характерным для всех команд сравнения является формирование не единственного результата (1 — истина, 0 — ложь), как в классической системе команд, а *пары* значений — результата и его отрицания, которые записываются в пару соседних предикатных (однобитовых) регистров. Эта пара значений связывается, скажем, с двумя параллельно выполняющимися ветвями алгоритма. Значения в предикатных регистрах определяют, результаты выполнения какой из ветвей должны быть выбраны для дальнейших действий.

Отметим, что значения из предикатных регистров могут использоваться не только в командах условного перехода, как в обычных командах процессора. Почти все команды системы команд IA64 могут использовать эти регистры. Такой подход позволяет во многих случаях избежать применения условных переходов, которые отрицательно сказываются на производительности процессоров. Вместо организации ветвления процессор, имеющий большое число ресурсов, в частности, регистров и функциональных блоков, может *исполнить обе ветви программы одновременно.*

В качестве примера предикатной организации ветвления рассмотрим оператор из программы на языке Паскаль: `if i=j then k:=k+1 else m:=m+1`. Пусть значения величин `i` и `j` находятся в регистрах `r4` и `r5`, а переменных `k` и `m` — в регистрах `r6` и `r7` соответственно. Тогда на Ассемблере процессора, относящегося к архитектуре IA64, этот оператор может быть записан в виде следующего фрагмента:

```
cmp.eq r4,r5,p3
 <p3> add r6,1,r0
 <p4> add r7,1,r0
```

Мнемокод первой команды, `cmp.eq`, показывает, что должно быть выполнено сравнение величин `i` и `j` в операции равенства. Первый и второй параметры команды, `r4` и `r5`, задают регистры, в которых находятся сравниваемые коды. Третий параметр команды, `p3`, задает один из предикатных регистров, в который

должен быть записан результат сравнения. Отрицание этого результата автоматически записывается в парный к p3 регистр p4. Таким образом, по первой команде фрагмента cmp.eq сравниваются значения i и j и результат заносится в пару взаимных предикатных регистров p3 и p4.

Две следующие команды сложения, add r6, 1, r0 и add r7, 1, r0, осуществляют сложение значения из регистра r6 или r7 с непосредственным операндом 1, а результат направляется в регистр r0. Из их записи видно, что они используются с указанными предикатами p3 и p4. Если значения i и j совпадают, то значения, попадающие в предикатные регистры p3, p4, равны 1 и 0 соответственно. Это означает, что результат выполнения первой команды add должен учитываться, а результат выполнения второй — нет. Обе команды add выполняются параллельно, и только в самом конце конвейера происходит проверка того, истинно ли предикатное условие команды, и в зависимости от этого производится запись в выходной регистр r0 того или иного результата. При этом может использоваться подмена регистров. Преимущество такого подхода состоит в линейном характере выполнения этого участка программы. Таким образом, предикатность команд избавляет от необходимости организовывать ветвление, прогнозировать переход и останавливать конвейер.

В 1990–1993 гг. совместными усилиями компаний Intel и HP (Hewlett-Packard — название фирмы образовано от фамилий основателей компании) было начато проектирование первого 64-битового процессора. Проект базировался на предшествующих разработках архитектуры с названием PA Wide Word (от Parallel Architecture Wide Word — всемирная параллельная архитектура). Немного позже эту архитектуру стали называть SP PA (от Super Parallel Processor Architecture — суперпараллельная процессорная архитектура), а затем она получила современное название EPIC. Конкретной реализацией ее и является обсуждаемая архитектура IA64.

В 2001 г. разработчиками компании Intel по 0,18-микронной технологии был выпущен первый процессор Merced с указанной архитектурой, а в 2002 г. в компании HP был создан процессор второго поколения с названием McKinley. Впоследствии этим реализациям дали официальное название Itanium.

В 2003 г. были выпущены микропроцессоры Itanium 2 с архитектурой IA-64 и тактовыми частотами 1,4 и 1,5 ГГц, выполненные по 0,13-микронной технологии. Максимальный объем оперативной памяти в системах на базе Itanium 2 может достигать 2 Тбайт. Предусмотрена возможность создания многопроцессорных систем, которые могут включать до 128 процессоров. В таких системах обеспечен высокоскоростной обмен данными между процессорами, устройствами ввода/вывода и памятью по 128-разрядной системной шине с пиковой пропускной способностью 6,4 Гбайт/с.

Считается, что процессоры моделей Itanium и Itanium 2 образуют семейство **IPF** (от Itanium Processor Family — семейство процессоров Itanium). В 2005 г. это семейство IPF пополнилось моделью Itanium 2 Processor Enhancement, выполненной по 90-нанометровой технологии с частотой более 1,6 ГГц, кэшем третьего уровня L3 объемом 9 Мбайт и кэшем четвертого уровня с объемом 32 Мбайт.

Следует отметить, что Itanium — это двухрежимные процессоры, которые могут работать как в 32-битовом, так и в 64-битовом режимах. То есть 64-битовые процессоры обеспечивают возможность аппаратной эмуляции 32-битовой системы команд x86. В настоящее время производительность процессоров IPF на программах, написанных в системе команд x86, меньше, чем производительность процессоров, относящихся к архитектуре IA32, так как на эмуляцию тратится дополнительное время.

В соответствии с планами компании Hewlett-Packard по развитию вычислительных систем, в настоящее время Itanium является стратегическим, единым процессором, и на него постепенно будут переводиться все аппаратные платформы, которыми обладает компания.

16.1.8. Семейства, программно совместимые с моделями Intel

Как отмечалось ранее, компьютеры, построенные на базе процессоров Intel, весьма популярны в мире. Для них создается огромное количество всевозможных программ. Поэтому многие фирмы-производители вычислительной техники разрабатывают и выпускают свои модели компьютеров, программно совместимые с моделями процессоров Intel. Производством совместимых моделей процессоров заняты такие фирмы как AMD (от Advanced Micro Devices — улучшенные микроустройства), Cyrix, IBM, Texas Instruments и некоторые другие. Обычно эти фирмы выпускают совместимые модели немного позже, чем это делает Intel, но они включают в разработки только отработанные, показавшие свою эффективность архитектурные решения. За счет улучшенных технических подходов им удается увеличить производительность либо уменьшить стоимость аналогичной модели по сравнению с оригинальной моделью из семейства Intel. Конкуренты Intel предлагают альтернативные варианты, для которых требуется минимальное количество новых команд или не требуется переработка компиляторов, что дает им некоторые преимущества.

В качестве примера программно совместимых моделей рассмотрим процессоры, выпущенные фирмой AMD. Первой получившей признание моделью фирмы AMD были процессоры AMD Krypton 5, или, в краткой форме, AMD K5. Эти процессоры построены по архитектуре x86-to-RISC86 и полностью совместимы с процессором Pentium, чьим соперником они считались. Процессор AMD K5 работал на частотах от 90 до 133 МГц и содержал 16 Кбайт кэш-памяти для кодов команд и 8 Кбайт для кодов данных.

Первые экземпляры следующей модели, AMD K6, производились по 0,35-микронной технологии, работали на частоте 166 МГц и имели раздельный кэш из двух блоков объемом по 32 Кбайт каждый. Это суперскалярные процессоры, полностью совместимые с командами системы x86 на уровне двоичных кодов. В систему команд процессора была включена поддержка набора целочисленных машинных команд группы MMX, что делало процессор AMD K6 сопоставимым

с процессором Pentium II фирмы Intel. Вместе с тем производительность процессора на операциях с плавающей точкой была невысока.

Затем были выпущены процессоры AMD K6-2 и AMD K6-3, изготавливавшиеся по 0,25-микронной технологии и работавшие на частотах от 266 до 450 МГц. Эти процессоры уже составляли серьезную конкуренцию модели Pentium II. Например, AMD K6-2 с тактовой частотой 333 МГц имел пиковую производительность 1,33 Гфлоп, в то время как Pentium II с частотой 400 МГц показывал только 0,4 Гфлоп. Преимущество объясняется, в частности, тем, что для этих процессоров был разработан фирменный набор машинных команд с названием 3DNow!, который оказался более эффективным заменителем команд группы SSE процессора Pentium II. Так, например, группа команд 3DNow! включает 21 машинную команду, в то время как группа SSE — 71 команду.

Следующей известной моделью фирмы AMD стали процессоры седьмого поколения Athlon K7. Они производились по 0,25-микронной технологии, имели тактовые частоты от 500 до 700 МГц, внутренний кэш объемом 128 Кбайт и показатели 31,7 SPECint95 и 24,0 SPECfp95 на частоте 700 МГц. Последние модели AMD K7 производились уже по 0,18-микронной технологии и работали на частоте 1,4 ГГц. Отличительной особенностью этой модели стала 43-битовая адресная шина. Это позволило адресовать до 8 Тбайт оперативной памяти, что существенно выше, чем 64 Гбайт у процессора Pentium III. Процессор Athlon K7 относится к суперскалярным суперконвейерным. Он имеет 9 функциональных исполнительных устройств. В соответствии с этим Athlon K7 может выполнять до 9 команд за такт, что тоже больше, чем у Pentium III. В архитектуре Athlon K7, как и в архитектуре Pentium, предусмотрено внеочередное выполнение команд. Система команд процессора Athlon K7 поддерживала группу команд MMX процессора Pentium III. Кроме того, в группу команд 3DNow! были включены 24 новые команды. Получившуюся расширенную группу стали называть Enhanced 3DNow!.

К этому же поколению относится и выпущенный немного позднее 32-битовый процессор AMD Athlon XP. Он изготавливается по 0,13-микронной технологии, имеет тактовую частоту до 1,8 ГГц, внутренний кэш объемом 384 Кбайт и возможность работы с шиной 400 МГц.

К восьмому поколению процессоров фирмы AMD относится уже 64-разрядный процессор AMD Athlon 64 с тактовой частотой 2,2 ГГц. Он имеет специальный набор команд типа x86-64, обеспечивающий работу как с 32-, так и с 64-битовыми приложениями. Важным архитектурным отличием процессоров AMD Athlon 64 является обращение к оперативной памяти напрямую, с использованием интегрированного с процессором контроллера памяти, а не через системную шину, как это делает процессор Athlon XP.

В 2005 г. фирма AMD выпустила состоящий из 233 млн. транзисторов двухъядерный процессор Athlon 64 X2. Последний компонент наименования отражает тот факт, что процессор имеет два ядра, выполненные по архитектуре AMD 64. Каждое ядро имеет собственную кэш-память второго уровня объемом 1 Мбайт и работает на частоте до 2,4 ГГц.

Сравнивая между собой модели фирмы Intel и AMD, а также других производителей программно совместимых процессоров, нельзя не заметить, что все фирмы используют схожие, а во многих случаях просто одинаковые архитектурные решения.

16.2. Семейство SUN SPARC

В 1981 г. Э. Бехтольсхайм для работы в сети Стэнфордского университета под управлением операционной системы Unix создал рабочую станцию, которая была им названа **SUN** (от Stanford University Network — сеть Стэнфордского университета). А в 1982 г. он стал одним из основателей компании Sun Microsystems, ныне одной из ведущих фирм, работающих в сфере информационных технологий.

В 1987 г. компания Sun разработала 32-битовый микропроцессор **SPARC** (от Scalable Processor ARChitecture — масштабируемая или наращиваемая архитектура процессора), который был одним из первых процессоров, построенных по RISC-архитектуре. Его система команд содержала всего 55 команд трех различных форматов. К 1995 г. в компании Sun на принципах RISC-архитектуры была разработана 64-битовая система команд SPARC v.9. Не ее базе была создана рабочая станция Sun Ultra SPARC I, предназначенная для работы с мультимедийной информацией, а также в качестве веб-сервера и в многопроцессорных системах. Затем было разработано несколько моделей семейства, в которых наращивалась производительность системы. Относительно недавно был выпущен двухъядерный процессор Sun UltraSPARC IV, в котором используется технология Chip Multithreading, незначительно отличающаяся от технологии Hyper Threading процессоров Pentium. Компьютеры семейства SPARC производятся фирмами Sun, Fujitsu Siemens Computers (FSC), Texas Instruments и некоторыми другими.

В качестве примера рассмотрим характеристики 64-разрядного суперскалярного процессора Sun UltraSPARC III. Он выполнен по 0,18-микронной технологии, состоит из 29 млн. транзисторов, имеет тактовую частоту до 900 МГц, внешний кэш объемом до 8 Мбайт, оперативную память объемом до 2 Тбайт и показатели 467 SPECint2000 и 483 SPECfp2000 на частоте 900 МГц.

Процессор UltraSPARC III содержит четыре целочисленных и два вещественных функциональных устройства, конвейер длиной 14 ступеней, четыре внутренних четырехвходовых кэша: кэш данных объемом 64 Кбайт, кэш команд объемом 32 Кбайт, а также кэш предварительной выборки и кэш записи объемом 2 Кбайт каждый.

Работающая на частоте 200 МГц шина внешнего кэша имеет разрядность 288 бит, из которых 256 бит информационных, а остальные 32 бита — контрольные, относящиеся к системе ECC. Соответственно, производительность этой шины равна 6,4 Гбайт (32 байта · 200 МГц).

Шина данных имеет разрядность 144 бита, из которых 128 информационных и 14 контрольных системы ECC. Частота шины равна 150 МГц, а ее пропускная способность составляет 2,4 Гбайт/с. Адресная шина с разрядностью 43 бита также имеет тактовую частоту 150 МГц.

Архитектура процессора позволяет запускать на исполнение до 6 команд одновременно, из них четыре целочисленных и две команды с плавающей точкой. Фактически исполнение целочисленных команд занимает всего один такт. Тем не менее используется задержка схода таких команд с конвейера на четыре такта. Такая задержка необходима для сохранения последовательности результатов в связи с тем, что у процессора UltraSPARC III отсутствует внеочередное исполнение — команды программы запускаются на конвейер и сходят с него в том порядке, в котором они заданы в программе. После выборки команды попадают в очередь, содержащую 20 элементов, откуда они группами, содержащими до шести команд, направляются в соответствующие исполнительные устройства. Все команды в группе получают собственный идентификационный код, в соответствии с которым на выходе из конвейера фиксируются результаты их выполнения. Описанный пакетный режим для команд довольно эффективно загружает функциональные устройства процессора даже при отсутствии переопределения порядка команд.

Важнейшим архитектурным отличием семейства SUN SPARC является явная нацеленность на обеспечение максимально возможной масштабируемости, которая позволяет объединить до нескольких сотен процессоров в единую вычислительную систему. В связи с этим критической точкой архитектуры стало соединение большого количества процессоров с системной шиной. Очевидно, что размещение на одной шине даже нескольких десятков слотов представляет сложно реализуемую задачу. В связи с этим было принято решение выделить каждому процессору индивидуальную шину, которые подсоединяются к центральному коммутатору, обеспечивающему необходимый обмен данными.

ВНИМАНИЕ

Коммутатором называется устройство, обеспечивающее путем необходимых внутренних переключений связь между любыми двумя устройствами, входящими в группу устройств, подключенных к коммутатору.

Понятно, что в этом случае «узким» местом архитектуры становится сам коммутатор. Однако разработчикам удалось создать коммутационные устройства, производительность которых обеспечивала все потребности архитектуры. Базовым элементом архитектуры SUN SPARC стал высокоскоростной коммутатор UPA (от Ultra Port Architecture). В связи с этим говорят, что компьютеры семейства SUN SPARC построены по архитектуре UPA.

ВНИМАНИЕ

Архитектура UPA — это высокопроизводительная многопортовая (многовходовая) масштабируемая архитектура, построенная на принципе одновременной пакетной коммутации нескольких портов и предназначенная для создания вычислительных систем с произвольным количеством процессоров.

Еще одной принципиальной особенностью архитектуры SUN SPARC является независимая коммутация адресов и данных. Такая особенность позволяет строить

системы с произвольной разрядностью шин адресов и данных для каждого канала связи с процессором. При этом передача адреса производится параллельно с прохождением предыдущего пакета данных.

Кратко обсудим особенности системы RISC-команд процессоров SPARC. Один процессор системы содержит:

- тридцать два 64-битовых регистра целых чисел, которые обозначаются `%r0`, `%r1`, ..., `%r31`;
- тридцать два 80-битовых регистра с плавающей точкой `%f0`, `%f1`, ..., `%f31`;
- двадцать регистров специального назначения, например: регистр `%PC` содержит адрес текущей команды, `%NPC` — адрес следующей команды и т. д.

Отметим, что некоторые из целочисленных регистров также имеют жестко фиксированное назначение: например, регистр `%r0` всегда содержит константу 0, а регистр `%r14` — указатель на вершину стека.

В системе команд семейства SPARC предусмотрено три формата команд, трехадресные обрабатывающие команды, использование только простых способов адресации (регистровая, непосредственная и прямая), а также автоматическое использование стека для хранения параметров процедур. Все команды имеют длину 32 бита, каждая команда выполняет только одно действие. Большинство команд имеет два входных регистра и один выходной. Вместо одного из регистров может указываться непосредственный операнд.

Для иллюстрации особенностей системы команд SPARC рассмотрим некоторые примеры. По команде `ldsb a, %r1` байт оперативной памяти с адресом `a`, содержащий знаковый код, копируется (загружается) в регистр `%r1` процессора, а по команде `ldub a,%r1` производится то же самое действие, но считается, что байт памяти содержит беззнаковый код. Команда `ldsh a,%r1` предписывает переслать в регистр `%r1` уже два байта из поля с адресом `a`. По команде `ldsw` выполняется загрузка слова, а по команде `ldd` — загрузка двойного слова. Команда `sth %r1, a` обеспечивает запись двух байтов из регистра `%r1` в поле памяти с адресом `a`.

Анализируя эти команды, можно заметить, что, в отличие от рассмотренной ранее системы команд процессоров x86, в системе команд процессора SPARC мнемокод команды явно задает длину операнда и формат обрабатываемого кода. Результат выполнения команды всегда помещается в последний операнд команды.

Еще одна особенность системы команд SPARC состоит в том, что стандартная обрабатывающая команда, например `add %r1, %r2, %r3`, задающая сложение содержимого регистров `%r1` и `%r2` и запись результата в регистр `%r3`, не формирует значения флажков. Чтобы сформировать эти значения, нужно использовать явное указание в мнемокоде: команда `addcc %r1, %r2, %r3` выполняет то же самое сложение, но при этом еще и устанавливаются характеризующие результат флажки.

Приведем еще ряд примеров, показывающих, как в системе команд SPARC задаются более сложные адреса операндов:

```
add %r9, 4, %r9! %r9:=%r9+4
ldb [%r8+78], %r9! Загрузить в %r9 значение из адреса [%r8+78]
stb %r10, [%r6-5]! Записать по адресу [%r-6] значение из %r10
```

Приведенные ранее примеры записаны по правилам, принятым в языке Ассемблер процессора SPARC. Легко заметить, что отличия от правил записи программ на Ассемблере процессора Intel есть, но они незначительны.

16.3. Семейства PA-RISC, Alpha, Power PC, MIPS

Начиная с 1986 г. фирма Hewlett-Packard Microprocessors выпускает процессоры семейства PA-RISC (от Precision Architecture RISC). Основное отличие моделей этого семейства — высокая надежность и точность вычислений, которая достигается благодаря повсеместному использованию систем контроля и кодирования, обеспечивающих регистрацию и исправление ошибок (ECC). В настоящее время выпускаются входящие в это семейство процессоры серии PA 8x00. Например, выполненный по 0,18-микронной технологии 64-битовый суперскалярный процессор PA-8700 состоит из 186 млн. транзисторов, имеет площадь 304 мм^2 и работает на частоте 800 МГц. В этом семействе, так же как и в семействе Intel, имеется двухъядерный процессор PA 8800. Он выполнен по 0,13-микронной технологии и состоит из 300 млн. транзисторов. Каждое ядро работает на частоте 1 ГГц и имеет собственную внутреннюю кэш память объемом 1,5 Мбайт. В систему входит общий для процессорных ядер кэш второго уровня объемом 32 Мбайт. Совместное использование ядрами внешнего кэша позволяет существенно повысить производительность и минимизировать задержки, связанные с обращением к оперативной памяти.

Перед разработчиками подразделения Digital Alpha Microprocessors фирмы DEC (от Digital Equipment Corporation), так же как и перед разработчиками фирмы Sun, стояла задача создания процессоров максимальной масштабируемости и производительности, чтобы обеспечить возможность производства на их основе высокопроизводительных многопроцессорных систем. К началу 90-х гг. XX в. был разработан удовлетворявший этим условиям 64-битовый RISK-процессор Alpha, который стал родоначальником семейства. Наиболее мощными моделями семейства являются выполненные по 0,18-микронной технологии процессор Alpha 21364 (или EV7) с тактовой частотой 1 ГГц и процессор Alpha 21464 (или EV8) с тактовой частотой до 2 ГГц. Впоследствии фирма DEC перешла к компании Compaq, а затем стала составной частью фирмы Hewlett-Packard, которая недавно объявила о намерении в недалеком будущем прекратить производство моделей PA-RISC и Alpha и перейти к производству унифицированной модели семейства Itanium, относящейся к EPIC-архитектуре.

Компаниями IBM, Motorola и Apple Computer в середине 1990-х гг. был разработан процессор RISC-архитектуры PowerPC. Основными направлениями эволюции в семействе Power PC стали устранение команд, которые препятствуют повышению тактовой частоты, и обеспечение длительного времени жизни архитектуры путем ее расширения до 64-битовой. Кроме того, с одной стороны, осуществлялось упрощение архитектуры с целью ее приспособления для реализации

дешевых однопроцессорных систем, а с другой — добавление свойств, необходимых для поддержки многопроцессорных систем. На более поздних этапах выполнялись работы по подключению технологий мультимедиа и распознавания речи. Наиболее мощными моделями этого семейства на сегодня являются выполненный по 0,13-микронной технологии процессор Power PC 970 (IBM Power 4) с тактовой частотой 1,8 ГГц и Power PC 970FX, который выполнен уже на основе 90-нанометровой технологии и работает на тактовой частоте 2 ГГц.

Основным направлением разработок фирмы SGI (от Silicon Graphics International) является создание сверхмощных многопроцессорных вычислительных систем. Для таких систем фирма, в частности, разработала собственный суперскалярный RISC-процессор MIPS (от Million Instructions Per Second). Наиболее производительной моделью семейства на сегодняшний день является MIPS R14000.

Таблица 16.2. Результаты тестирования некоторых моделей процессоров

Модель процессора	Частота, ГГц	SPECint2000	SPECfp2000
HP Alpha 21364	1,15	877/795	1482/1124
HP PA-8700	0,875	678/642	647/600
Intel Itanium 2	1,0	810	1431
Intel Pentium EE	3,066	1130/1085	1103/1092
AMD Athlon XP 2800	2,25	933/898	843/782
IBM Power 4	1,45	935/909	1295/1221

В табл. 16.2 представлены официальные результаты выполненного в 2004 г. тестирования группы наиболее высокопроизводительных современных микропроцессоров на тестах SPECcpu2000. В числителе представлена пиковая производительность, а в знаменателе — реально показанная в тестировании. Для процессора Itanium 2 приведены только результаты тестирования. Анализ результатов тестирования показывает, что представленные в ней процессоры оказываются довольно близкими по своим показателям. Если же учесть, что тактовая частота процессора Intel Itanium 2, выбранного для тестирования, равна всего 1 ГГц, можно, по-видимому, по результатам проведенного тестирования отдать предпочтение этому процессору.

16.4. Семейства БЭСМ и Эльбрус

До сих пор обсуждались в основном вычислительные системы зарубежных разработчиков, а достижения отечественных ученых и конструкторов оставались в тени. Вместе с тем идеи и разработки советских и российских специалистов играют немаловажную роль в общемировом развитии информационных технологий.

Как уже упоминалась, первая советская электронная вычислительная машина МЭСМ, разработанная под руководством академика С. А. Лебедева, была выпущена в 1951 г. — всего на пять лет позже первой в мире электронной машины

ENIAC и на два года позже первой машины с хранимой программой EDSAC. Разработка этой машины началась в 1947 г. — практически сразу же после Великой Отечественной войны. Следует учесть, что с 1941 г. страна воевала и с 1945 г. восстанавливала разрушенное во время войны, а разработка вычислительных машин требует немалой концентрации финансовых средств, технологических решений и высококвалифицированных кадров. Поэтому выпуск работающей машины уже в 1951 г. можно считать серьезным достижением отечественной науки и техники.

Машина МЭСМ работала со скоростью порядка 100 операций в секунду. А уже в 1953 г. тем же коллективом разработчиков была создана родоначальница знаменитого отечественного семейства машина БЭСМ, быстродействие которой составляло 10 000 операций в секунду — самый лучший в Европе и очень неплохой в мировом масштабе показатель.

В 1954 г. советский математик А. А. Марков предложил теорию нормальных алгоритмов, которая позже была названа в его честь. Эта теория наряду с работами Тьюринга и Поста заложила основы теоретического программирования, построения трансляторов и операционных систем.

В 1957 г. на ежегодной сессии Академии наук СССР академик С. А. Лебедев сделал доклад, в котором он изложил основные идеи развития архитектуры вычислительных систем на длительную перспективу (см. 5.3.3).

В 1966 г. коллективом специалистов во главе с академиком С. А. Лебедевым и его заместителями В. А. Мельниковым и Л. Н. Королевым была закончена разработка машины БЭСМ 6, которая занимает особое место в развитии отечественной вычислительной техники. Это была полупроводниковая машина с тактовой частотой 10 МГц и быстродействием до 1 млн. операций с плавающей точкой в секунду. По своим показателям она входила в группу лучших в мире вычислительных систем. Хотя машину БЭСМ 6 обычно относят ко второму поколению, по архитектурным решениям она сопоставима с передовыми компьютерами третьего поколения.

После выпуска БЭСМ 6 было начато проектирование значительно превосходившей ее по своим характеристикам машины БЭСМ 10. Кроме того, к 1967 г. другой коллектив под руководством М. А. Карцева уже разработал проект полностью параллельной машины М-9 с фантастической по тем временам скоростью 1 млрд. операций в секунду. Но этим проектам не суждено было воплотиться в жизнь, так как руководством страны было принято глубоко ошибочное решение сконцентрировать все финансовые средства и кадровые резервы на копировании популярного в то время семейства американских машин IBM/360. Это решение отбросило отечественную электронную промышленность на десятилетия назад.

Несмотря на возникшие трудности, коллектив специалистов Института точного машиностроения и вычислительной техники под руководством В. С. Бурцева и Б. А. Бабаяна продолжил развитие отечественных разработок, и к 1977 г. была создана многопроцессорная вычислительная система «Эльбрус 1» с производительностью до 10 Мфлоп (10 млн. операций в секунду). В 1984 г. была выпущена

система «Эльбрус 2» с производительностью 100 Мфлоп, а в 1991 г. была передана в эксплуатацию вычислительная система «Эльбрус 3.1» с производительность 400 Мфлоп. В 1987 г. за разработку и внедрение микропроцессорной системы «Эльбрус 2» один из руководителей проекта, Борис Арташесович Бабаян, был удостоен Ленинской премии.

Следует подчеркнуть, что в микропроцессорах западных производителей суперскалярный подход, аналогичный использованному в системе «Эльбрус 1», впервые был реализован только в 1991 г. А равноценный «Эльбрусу 1» суперскалярный процессор Pentium Pro фирмы Intel был создан еще позже — в 1995 г. Вот как об этом писал в журнале Microprocessor Report в феврале 1999 г. Кит Диффендорф, разработчик суперскалярного процессора Motorola 88110: «В 1978 году, почти на 15 лет раньше, чем появились первые западные суперскалярные процессоры, в „Эльбрус 1" использовался суперскалярный процессор с выдачей двух команд за один такт, изменением порядка исполнения команд, переименованием регистров и исполнением по предположению»[1]. По словам К. Диффендорфа, компьютеры «Эльбрус», в которых реализованы основные принципы современных архитектур, такие как симметричная многопроцессорная, суперскалярная и EPIC-архитектуры, были запущены в производство задолго до того, как побочные идеи начали только обсуждаться на Западе. Наряду с идеями академика С. А. Лебедева, которые широко использовались в разработках вычислительных систем во всем мире, сказанное свидетельствует о приоритете отечественных ученых и специалистов в разработке важнейших на современном этапе архитектурных решений, обеспечивающих существенное повышение производительности вычислительных систем.

Руководитель разработок моделей семейства «Эльбрус» Б. А. Бабаян с 1956 по 1996 г. работал в Институте точной механики и вычислительной техники Академии наук СССР (с 1992 г. — Российской академии наук). С 1993 г. он являлся научным руководителем московского центра SPARC. В связи с этим следует отметить, что значительную часть разработки процессора Sun UltraSPARC и операционной системы Sun Solaris выполнила группа отечественных специалистов под руководством Б. А. Бабаяна, которая с 1992 г. тесно сотрудничала с корпорацией Sun Microsystems. Достаточно обратить внимание на начало периода сотрудничества корпорации Sun с группой Б. А. Бабаяна и начало периода популярности машин семейства.

В этот же период Б. А. Бабаян являлся одним из руководителей российской компании «Эльбрус Интернэшнл», которая в 1999 г. опубликовала технические характеристики своей последней разработки — EPIC микропроцессора Эльбрус E2K (Эльбрус 2000). Оказалось, что этот процессор работает примерно в пять раз быстрее, чем современный ему 64-битовый процессор Merced фирмы Intel. Процессор E2K изготавливался по 0,18-микронной технологии, занимал площадь 126 мм^2, работал на частоте 1,2 ГГц и имел показатели 135 SPECint95 и 350 SPECfp95. В то же время процессор Merced площадью 300 мм^2 работал на частоте 800 МГц и имел показатели 45 SPECint95 и 70 SPECfp95.

---

[1] Dieffendorf K. Russians Are Coming // Microprocessor Report. — 1999. — Vol. 13, № 2. — P. 1.

Архитектурные новшества процессора E2K были столь же впечатляющие. Он реализовал лучшую схемотехнику, чем та, которая использовалась в наиболее мощном к 2000 г. процессоре Alpha 21264. Для процессора Эльбрус E2E был готов полный набор системного программного обеспечения, включавший в себя распараллеливающий транслятор. Этот компилятор достигал показателя 10 операций за такт, что почти в три раза выше, чем у лучшего зарубежного аналога этого периода. Следует отметить, что использованные в разработке семейства «Эльбрус» оригинальные технические решения защищены 70 патентами США.

Серийное производство микропроцессора E2K должно было начаться в 2001 г., но из-за отсутствия финансирования его массовое производство так и не было осуществлено. С августа 2004 г. Б. А. Бабаян работает в должности директора по архитектуре в подразделении программных решений корпорации Intel. Он является руководителем глобального проекта в области архитектуры вычислительных систем, технологии двоичной компиляции и технологи безопасных вычислений, направленных на борьбу с вирусами. Одновременно Б. А. Бабаян остается директором Института микропроцессорной техники РАН. По сообщению журнала «Компьютер пресс» от 1.01.2005, в ноябре 2004 г. он стал первым европейским ученым, удостоенным за заслуги в достижениях корпорации звания Intel Fellow (почетный член научного общества корпорации Intel), которое присвоено только 42 специалистам в мире.

В настоящее время в фирме Intel работает и другой бывший разработчик системы «Эльбрус» В. М. Пентковский. В 1970–1980-е гг. в Институте точного машиностроения и вычислительной техники В. М. Пентковский принимал участие в разработке суперкомпьютеров «Эльбрус 1» и «Эльбрус 2». А в 1986 г. он возглавил проект разработки 32-разрядного процессора Эль-90, в котором сочетались концепция RISC и архитектурные решения процессора «Эльбрус 2». К 1990 г. проект находился на стадии завершения, и появились первые образцы нового процессора. Но в 1992 г. финансирование разработок прекратилось, и В. М. Пентковский перешел на работу в фирму Intel, где стал ведущим разработчиком процессоров. По-видимому, следует напомнить, что именно в 1993 г. Intel представила свой принципиально новый 32-разрядный процессор Pentium, а к 1995 г. — более совершенный процессор Pentium Pro, который уже вплотную приблизился по своим возможностям к отечественному процессору «Эльбрус» образца 1990 г. Официально В. М. Пентковский в этот период являлся главным архитектором процессора Pentium III. Сравнивая архитектурные особенности и технические характеристики проекта Эль-90 и реализации процессора Pentium III, легко заметить огромное количество, скорее всего, не случайных совпадений.

Можно утверждать, что вместе с Б. А. Бабаяном и другими разработчиками системы «Эльбрус» в фирмы Sun и Intel пришли огромный опыт и самые совершенные отечественные технологии, разработанные в Институте точного машиностроения и вычислительной техники, но так и не реализованные в России. Анализируя развитие отечественной вычислительной техники, нельзя не заметить высокие достижения и огромный потенциал отечественных специалистов

и ту негативную роль, которую сыграли ошибочные управленческие и политические решения.

Дополнительный материал по обсуждавшимся в главе вопросам можно найти в изданиях [28], [30].

Контрольные вопросы и упражнения

1. Дайте общую характеристику семейству процессоров Intel.
2. Опишите начальный этап развития процессоров семейства Intel.
3. Охарактеризуйте 16-битовые модели семейства Intel. Какие новые архитектурные решения появились в моделях семейства?
4. Охарактеризуйте третье поколение семейства Intel. Какие новые архитектурные решения появились в этом поколении?
5. Охарактеризуйте четвертое поколение семейства Intel. Какие новые архитектурные решения появились в этом поколении?
6. Охарактеризуйте отличительные особенности микропроцессора Pentium.
7. Опишите особенности технологии SIMD. В каком поколении микропроцессоров Intel она впервые нашла применение?
8. Что понимается под архитектурой двойной независимой шины? Когда и в какой модели микропроцессоров Intel она была использована?
9. Что нового появилось в архитектуре шестого поколения микропроцессоров Intel?
10. Охарактеризуйте архитектуру микропроцессора Pentium 4.
11. Охарактеризуйте особенности технологий Hyper Threading и EMT64.
12. Опишите особенности двухъядерных моделей Intel.
13. Опишите недостатки архитектуры IA32.
14. Охарактеризуйте основные отличия архитектуры IA64.
15. Поясните смысл понятий «пучок» и «связка».
16. Как реализуется и что дает предикатное выполнение команд в архитектуре IA64?
17. Охарактеризуйте модели семейства IFP.
18. Охарактеризуйте семейство AMD и сравните его с семейством Intel.
19. Расскажите о семействе SUN SPARC. Опишите особенности модели SUN Ultra SPARC III.
20. В чем принципиальные различия семейств SUN SPARC и Intel?
21. Дайте характеристику семейств PA-RISC, Alpha, Power PC и MIPS.
22. Охарактеризуйте семейства БЭСМ и «Эльбрус».

Глава 17
Параллельные архитектуры

Существует класс практически важных задач, для решения которых возможностей однопроцессорных компьютеров не хватает. Это задачи метеопрогноза, проектирования авиационной и космической техники, обработки изображений, управления оборонительными системами и т. д. Во многих случаях кроме высокой производительности от вычислительной системы требуется постоянная готовность к выполнению ее функций и высокая надежность, обеспечивающая работоспособность даже в случае отказа некоторой части оборудования. Готовность к работе для однопроцессорных систем ограничена необходимостью проведения периодических профилактических работ, а возможный отказ единственного центрального процессора в любой момент может полностью парализовать работу всей системы. Таким образом, можно утверждать, что, несмотря на достигнутое благодаря современным архитектурным и технологическим подходам существенное увеличение тактовых частот и производительности, однопроцессорные системы перечисленным выше требованиям не удовлетворяют.

Высокая производительность, высокая степень готовности к работе и отказоустойчивость могут быть обеспечены только с помощью многократного дублирования основных узлов системы, в том числе с помощью включения в состав компьютера более одного процессора. Необходимо обеспечить реально параллельное, то есть *одновременное* выполнение двух или более частей одной и той же программы двумя или более процессорами на одном или нескольких связанных компьютерах.

ВНИМАНИЕ

Параллельным называется такое выполнение программы, когда две или более части одной и той же программы выполняются одновременно двумя или более центральными процессорами одного и того же или разных компьютеров. В первом случае архитектура вычислительной системы считается многопроцессорной (мультипроцессорной), а во втором — многомашинной (мультикомпьютерной). Вычислительные системы, содержащие более одного центрального процессора, называются параллельными.

Не следует путать параллельную работу с многопрограммным (мультипрограммным) режимом вычислительных систем, когда все имеющиеся ресурсы разделяются между двумя или более программами, находящимися на стадии выполнения. Многопрограммный режим может быть организован и на однопроцессорной системе.

17.1. Законы Амдала

Пусть пиковая производительность однопроцессорной системы равна T_1, тогда пиковая производительность системы, состоящей из p таких процессоров, увеличивается в p раз, $T = pT_1$. **Ускорением параллельной системы** R назовем отношение ее производительности к производительности соответствующей однопроцессорной системы. Тогда пиковое ускорение $R = T/T_1 = p$. Значит, пиковое ускорение p-процессорной системы равно p.

Реальная производительность параллельной системы зависит не только от количества процессоров в системе и их производительности. Существенным фактором, влияющим на реальное ускорение системы, оказываются свойства выполняемой программы. Американский специалист в области вычислительных систем Р. Амдал, исследовавший вопросы производительности параллельных систем, доказал несколько утверждений, которые принято называть **законами Амдала**. Приведем первые два закона без доказательств. Желающие могут найти эти доказательства в [11].

Первый закон Амдала. Производительность вычислительной системы, состоящей из нескольких связанных между собой устройств, определяется самым непроизводительным устройством.

Этот закон является частным случаем общего физического положения, в соответствии с которым надежность любой системы определяется самым ненадежным ее элементом. Практическое значение этого утверждения состоит в том, что для повышения производительности вычислительной системы не имеет смысла существенно увеличивать производительность только центрального процессора и при этом оставлять на прежнем уровне возможности его шин, памяти и дисковой подсистемы. Общая эффективность всегда окажется связанной с самым неэффективным компонентом.

Почти в любой программе можно выделить участки, которые допускают распараллеливание. Это означает, что на таком участке существуют ветви, которые могут быть переданы для одновременного выполнения нескольким центральным процессорам. Кроме того, в программах существуют участки, допускающие только последовательное выполнение операций. Пусть, например, нужно найти сумму $\overline{c} = \{c_1, c_2, \ldots, cn\}$ двух заданных векторов $\overline{a} = \{a_1, a_2, \ldots, an\}$ и $\overline{b} = \{b_1, b_2, \ldots, bn\}$, где $ci = ai + bi$, $1 \le i \le n$. Если в нашем распоряжении имеется n процессоров, то, передав каждому из них пару соответствующих компонентов ai и bi и организовав их одновременную работу, можно получить искомый результат — все компоненты вектора $\overline{c}$ за время выполнения одного сложения. Видно, что участок программы, связанный с получением компонент вектора $\overline{c}$, допускает распараллеливание.

Однако в этой же программе имеются участки (например, связанные с вводом исходных данных, с подготовкой к циклу, с выводом результатов), — которые сложно или же вообще невозможно распараллелить. Наличие таких участков в программах является фактором, ограничивающим реальное повышение производительности многопроцессорных систем. Количественно это ограничение описывается вторым законом Амдала.

Второй закон Амдала. Пусть вычислительная система состоит из p процессоров. Предположим, что k из N операций алгоритма могут выполняться только последовательно. Пусть $\beta = k/N$ — доля последовательных операций в алгоритме, $0 \le \beta \le 1$. Тогда максимально возможное ускорение системы

$$R = \frac{1}{\beta + (1-\beta)/p}.$$

Очевидно, что предел выражения для ускорения R при неограниченном увеличении количества процессоров, то есть при p, стремящемся к бесконечности, равен $1/\beta$. Пусть, например, доля операций, которые могут быть выполнены только последовательно, равна $\beta = 0{,}1$, тогда реальное ускорение R не может быть больше 10 при любом количестве процессоров. Практический вывод из этих соображений состоит в том, что для общего повышения эффективности нужно не только наращивать количество процессоров в системе, но и улучшать свойства программы, в частности, уменьшать долю операций, выполняемых только последовательно.

17.2. Топология параллельных систем

Из-за наличия в составе параллельной вычислительной системы более одного центрального процессора его уникальность в составе системы теряется. Процессор становится одним из многих аналогичных узлов. Поэтому в параллельных вычислительных системах термин «центральный процессор» заменяется термином **процессорный элемент** (ПЭ), **процессорный модуль** (ПМ) или **процессорный узел** (ПУ).

ВНИМАНИЕ

Процессорным элементом считается любой процессор в составе параллельной вычислительной системы.

Вычислительные системы с параллельной архитектурой характеризуются:

- типом и мощностью процессорных элементов;
- масштабируемостью, то есть возможным количеством процессорных элементов;
- объемом и типом модулей памяти;
- возможными связями и способом взаимодействия между процессорными элементами, а также между процессорными элементами и модулями памяти.

В современных параллельных вычислительных системах используются высокопроизводительные, в основном 64-битов, процессоры PA-RISC (PA 8700, PA 8800), Alpha (Alpha 21264, Alpha 21364, Alpha 21464), Power PC (Power 3, Power 970, Power 4), MIPS (R14000, R16000), Sun (UltraSPARC III, UltraSPARC IV), AMD (Opteron, Athlon 64), Intel (Xeon EM64T, Itanium 2) и некоторые другие.

Различные системы допускают различную масштабируемость. Характерным является возможность включения в систему от 128 до 512 процессорных элементов. Вместе с тем существуют системы, содержащие десятки и сотни тысяч процессоров.

Оперативная память в современных параллельных системах реализуется на высокопроизводительных микросхемах типа DDR SDRAM. Ее общий объем обычно составляет десятки терабайт. Каждый процессор может иметь собственный внутренний кэш. Кроме того, в систему обычно включаются общие или индивидуальные внешние кэши нескольких уровней.

Основными элементами структуры вычислительной системы, влияющими на ее производительность, являются способ соединения процессорных элементов и модулей памяти, а также организация связи между ними. Эти факторы оказываются даже более весомыми, чем индивидуальная мощность процессорных элементов, из которых построена система.

ВНИМАНИЕ

Способ соединения процессорных элементов и модулей памяти в параллельной вычислительной системе называется топологией системы.

Существуют **статические** и **динамические** топологии. В статических топологиях соединение элементов системы фиксировано и не изменяется с течением времени. В динамических схемах все компоненты системы подключаются к переключательному устройству — **коммутатору**, которое может соединять любые компоненты друг с другом. В целом статические топологии отличаются, с одной стороны, более высоким быстродействием, так как все связи между процессорными элементами известны и уже установлены, а с другой — невысокими возможностями масштабирования системы, так как включение нового элемента обычно требует физической перестройки системы связей и перенастройки программного обеспечения. Динамические топологии, наоборот, отличаются высокими возможностями масштабируемости и меньшим быстродействием.

Существует много различных способов построения коммутаторов, используемых для организации связей внутри параллельных систем. На рис. 17.1 изображены два варианта их построения. В **матричном** коммутаторе (рис. 17.1, *а*) каждый процессорный элемент имеет линии связи, которые связывают его с каждым из модулей памяти (МП). В узлах пересечения линий связи находятся простые переключающие устройства, обеспечивающие соединение или разрыв связи. В общем случае такой коммутатор может содержать $n \times m$ входов, к которым могут подключаться процессорные элементы, модули памяти или другие коммутаторы в любых нужных комбинациях. Достоинствами такого устройства коммутатора являются его высокое быстродействие, возможность одновременной связи любых

процессорных элементов между собой и с любыми модулями памяти, высокая надежность. Но матричные коммутаторы имеют существенный недостаток — количество необходимого оборудования растет как n^2 при увеличении количества входов n.

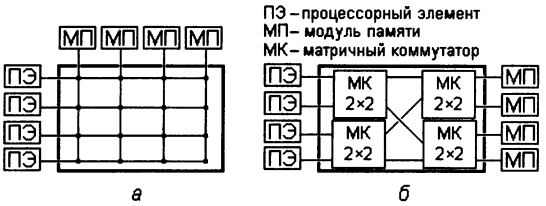


Рис. 17.1. Упрощенные схемы коммутаторов: *а* — матричного; *б* — каскадного

Для удешевления коммутаторов используются различные **каскадные** схемы. На рис. 17.1, *б* показана схема коммутатора, построенного на базе простых матричных коммутаторов (МК 2 × 2), имеющих 2 × 2 входа. Каждый из простых коммутаторов связан с входом другого простого коммутатора, а также с двумя процессорными элементами или модулями памяти. Так обеспечивается связь между любыми двумя подключенными к коммутатору устройствами, аналогичная связи в матричных коммутаторах. При этом общее количество переключательных элементов в такой схеме возрастает только как $(n \log_2 n)/2$, что для больших n гораздо меньше, чем n^2. Однако наличие промежуточных коммутаторов вызывает замедление работы устройства в целом.

Топология вычислительной системы может быть изображена в виде графа, узлами которого являются процессорные элементы, модули памяти или коммутаторы. Ребра такого графа отображают существующие между элементами системы связи.

Топология, в которой каждый узел имеет связь с любым другим (рис. 17.2, *а*), называется **полносвязной**. Такая топология отличается наиболее высокой производительностью, но имеет ограниченные возможности масштабирования и высокую стоимость. Если система содержит n узлов, то каждый узел должен иметь $n - 1$ связей, а общее количество связей в полносвязной топологии равно $n(n - 1)/2$. В большинстве случаев разработчики не имеют материальных ресурсов для построения таких систем.

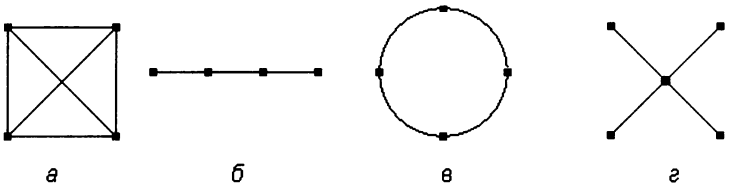


Рис. 17.2. Базовые топологии параллельных вычислительных систем

Если в топологии отсутствует связь между какими-либо двумя узлами системы, то такая топология считается **неполносвязной**. Существует множество вариантов неполносвязных топологий. На рис. 17.2, *б–г* изображены базовые неполносвязные

топологии параллельных вычислительных систем, из которых в конечном счете формируются любые используемые на практике топологии.

На рис. 17.2, *б* изображена топология с общей шиной, которая является аналогом архитектуры с общей шиной однопроцессорных компьютеров. Топология отличается простотой и дешевизной. Вместе с тем ей присуща низкая надежность, так как отказ любого узла приводит к выходу из строя всей системы. Кроме того, в топологии с общей шиной включение каждого нового узла приводит к уменьшению общей пропускной способности шины, что свидетельствует о плохой масштабируемости такой топологии.

На рис. 17.2, *в* представлена топология типа **кольцо**, являющаяся развитием топологии с общей шиной. Фактически кольцо представляет собой общую шину с соединенными концами, поэтому оно обладает теми же достоинствами и недостатками, что и общая шина. К преимуществам относится более высокая скорость обмена, так как передача данных может осуществляться в двух направлениях и можно выбрать более короткий путь, чем в случае использования общей шины.

Еще одна базовая топология — типа **звезда** — изображена на рис. 17.2, *г*. В системе выделяется один центральный узел, который имеет отдельную связь со всеми остальными узлами. Любой другой узел имеет связь только с центральным. Такая топология отличается в целом более высокой надежностью по сравнению с общей шиной или кольцом, но в ней предъявляются более высокие требования к производительности и надежности центрального узла, который становится критическим для системы.

Выбор топологических связей в конкретных системах обусловливается различными требованиями: стоимостью, технологичностью, надежностью, производительностью и т. д. В связи с этим к коммутаторам и построенным на их базе топологиям предъявляются противоречивые требования высокой пропускной способности, хорошей масштабируемости, приемлемой стоимости и т. д.

Существует огромное количество вариантов топологий, реализованных в вычислительных системах. Например, на рис. 17.3, *а* изображена топология типа двухмерной решетки, использованная в системе Intel Paragon. Ее естественным развитием является топология двухмерного тора (рис. 17.3, *б*), реализованная в вычислительных системах компании Dolphin Interconnection Solution. Переход от двухмерной решетки к двухмерному тору осуществляется аналогично переходу от общей шины к кольцу — замыканием каждой линии связи решетки в кольцо.

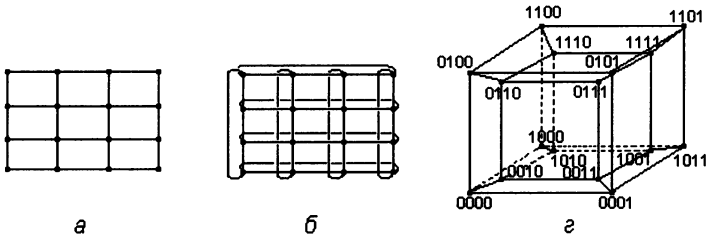


Рис. 17.3. Некоторые развитые топологии параллельных систем

На рис. 17.3, *в* приведен пример еще одной интересной топологии — двоичного *четырехмерного* гиперкуба. Двухмерный гиперкуб представляет собой обычный квадрат (содержащий четыре узла), а трехмерный — это обычный куб из восьми узлов. В общем случае в этой топологии строится состоящий из $2n$ узлов n-мерный гиперкуб, в котором каждый узел соединен с ближайшим узлом по каждому из n измерений. В двоичной системе номер любого из участвующих в связи узлов отличается от номера любого другого только в одном бите, что обеспечивает полную симметричность системы и простоту реализации множества задач вычислительной математики. Известны системы с этой топологией, содержащие до 65 536 узлов.

17.3. Классификация параллельных систем по Флинну

В настоящее время существует большое количество основанных на различных признаках и обеспечивающих различное разбиение классификаций параллельных вычислительных систем. Это классификации М. Флинна, Р. Хокни, Т. Фенга, В. Хендлера и некоторых других авторов. Подробный обзор этих классификационных схем можно найти в [11].

Классификационной схемой, которая получила признание у большинства специалистов и стала в некотором смысле базовой, является классификация, предложенная Майклом Флинном в 1966 г. Она основана на уже встречавшихся понятиях потока команд и потока данных. В связи с введением понятия процессорного элемента немного изменим их определения. Потоком команд будем называть последовательность команд программы, выполняемых отдельным процессорным элементом вычислительной системы. Потоком данных будем называть последовательность данных, вызываемых на обработку в один из процессорных элементов. Если количество одновременно выполняемых различными процессорными элементами вычислительной системы команд больше одной, то поток команд называется **множественным**. Если в вычислительной системе на одной и той же стадии обработки находится более одного набора операндов, которые поступают в различные процессорные элементы, то поток данных называется множественным.

Для обсуждения классификации Флинна нам потребуются еще некоторые понятия. Вычислительная система, состоящая из нескольких находящихся в одном или в разных помещениях компьютеров, которые имеют общую внешнюю память и общее программное обеспечение, называется многомашинной системой с **косвенной слабой связью** (рис. 17.4). Связь между компьютерами осуществляется через магнитные диски или магнитные ленты. Каждая машина работает под управлением собственной операционной системы. В архитектурах с косвенной связью обеспечиваются высокая готовность и надежность системы.

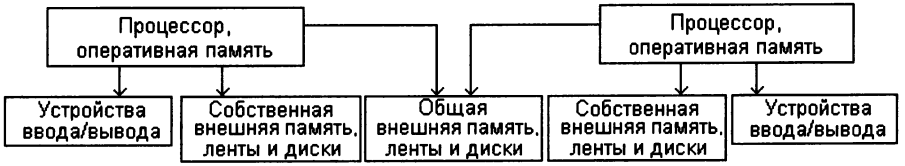


Рис. 17.4. Многомашинная вычислительная система с косвенной слабой связью

Вычислительная система, в которой компьютеры связаны через сетевые адаптеры и/или каналы ввода/вывода, называется многомашинной системой с **прямой слабой связью** (рис. 17.5). Каждая машина работает под управлением собственной операционной системы. Как правило, в архитектурах с прямой слабой связью одна из машин специализируется на обеспечении работы низкоскоростных внешних устройств, а вторая является высокоскоростным центральным вычислителем. Как косвенная, так и прямая слабая связь считаются **низкими** связями.

Рис. 17.5. Многомашинная вычислительная система с прямой слабой связью

Связь между двумя или более процессорными элементами вычислительной системы через общую оперативную память и, возможно, общее периферийное оборудование называется **сильной**, или **высокой** (рис. 17.6). Процессоры с сильной связью работают под управлением общей операционной системы. Сильная связь обеспечивает не только повышение готовности и надежности, но и увеличение производительности. Еще раз подчеркнем, что архитектура вычислительных систем со слабой связью считается *многомашинной*, а архитектура с сильной связью — *многопроцессорной*.

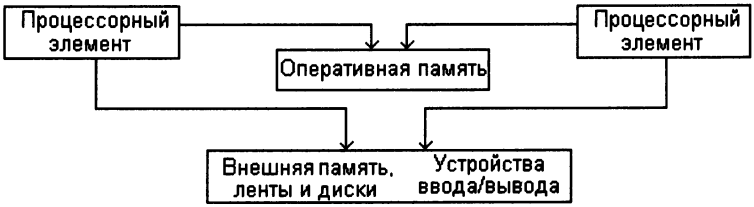


Рис. 17.6. Многопроцессорная вычислительная система с сильной связью

Классификация Флинна (рис. 17.7) основана на следующих признаках.

1. Одиночный (ОК) или множественный (МК) поток команд в центральной части вычислительной системы.
2. Одиночный (ОД) или множественный (МД) поток данных в центральной части вычислительной системы.
3. Пословный (С) или поразрядный (Р) способ обработки в центральной части вычислительной системы.

4. Низкая (Нс) или высокая (Вс) связность компонентов вычислительной системы.
5. Однородные (Ор) или неоднородные (Нр) основные компоненты вычислительной системы.
6. Тип внутренних связей в вычислительной системе: через внешнюю память (Пм), через канал ввода/вывода (Кн), связь типа «процессор-процессор» (Пр), по общей шине (Ош), через коммутатор (Пк).

Рис. 17.7. Фрагмент классификации Флинна

В классификации М. Флинна для определения классификационной группы вычислительной системы используется краткая формула, содержащая только одно обозначение из каждого уровня классификации. Обозначения размещаются в порядке от верхнего уровня к нижнему. После первых трех уровней записывается косая черта. Например, формула МКМДС/ВсОрПк обозначает вычислительную систему с множественным потоком команд и множественным потоком данных в центральной части, с пословной обработкой, высокой степенью связности (сильной связью), однородной структурой и связями между процессорными элементами и модулями памяти через коммутатор.

Существуют параллельные вычислительные системы, которые обладают признаками более чем одного класса. Такого рода системы принято считать системами с **комбинированной структурой**. Существуют также вычислительные системы, которые в процессе функционирования могут изменять режим работы, переходя при этом из одного класса в другой. Такие системы называются системами с **перенастраиваемой структурой**.

Некоторые признаки классификационной схемы Флинна в настоящее время практически не применяются. Поэтому на рис. 17.7 изображена только та часть этой классификации, которая используется и для современных вычислительных систем. Обсудим более детально архитектурные особенности основных групп вычислительных систем.

Архитектура **ОКОД**, или **SISD** (от Single Instruction Single Data stream) — одиночный поток команд и одиночный поток данных, — фактически представляет собой фон-неймановскую архитектуру однопроцессорных вычислительных систем

последовательного действия. Бытовым аналогом работы таких систем является ситуация, в которой один человек делает одну деталь, последовательно выполняя все необходимые операции по ее обработке.

В вычислительных системах типа ОКОД имеется только один поток команд (рис. 17.8, *а*). Все команды выполняются последовательно друг за другом, каждая команда инициирует выполнение процессорным элементом только одной скалярной операции. С середины 40-х и примерно по 70-е гг. XX в. этот класс вычислительных систем был практически единственным.

Разновидностями архитектуры ОКОД являются системы ОКОДС с пословной и ОКОДР с поразрядной обработкой данных.

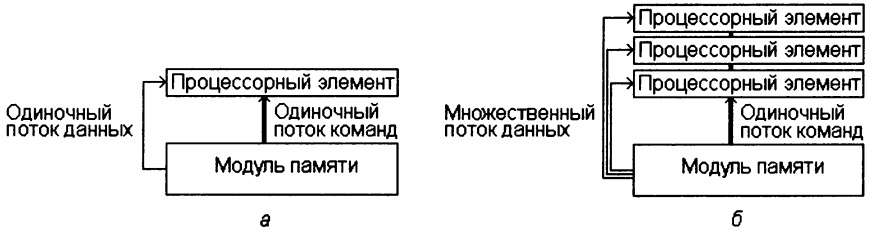


Рис. 17.8. Архитектуры с одиночным потоком команд: *а* — ОКОД; *б* — ОКМД

Архитектура **ОКМД**, или **SIMD** (от Single Instruction Multiply Data stream), — одиночный поток команд над множественным потоком данных. Бытовым аналогом работы таких систем является ситуация, в которой несколько человек одновременно, синхронно выполняют одну и ту же операцию, последовательность одинаковых операций над разными деталями. Примером также может служить ансамбль музыкальных инструментов, синхронно исполняющих мелодию под управлением дирижера.

В вычислительной системе с архитектурой ОКМД имеется единственное устройство управления, которое организует работу группы процессорных элементов. Это устройство управления создает только один поток команд, который может включать операции над векторными данными (рис. 17.8, *б*). Каждый процессорный элемент получает от устройства управления в каждый фиксированный момент времени одну и ту же команду и выполняет ее над своими (локальными) данными. Наличие нескольких процессорных элементов позволяет одновременно выполнять одно и то же действие над группой однотипных данных, например осуществлять одновременное сложение элементов двух векторов.

В связи с обсуждением этого класса параллельных систем, по-видимому, следует напомнить о включении основанных на подходе SIMD групп команд MMX и SSE (см. 16.1.4 и 16.1.5) в систему команд процессоров Pentium и аналогичных групп в системы команд других процессоров.

Основной разновидностью систем класса ОКМД являются системы ОКМДС, в которых выполняется пословная обработка данных. В этот класс входят векторные и матричные вычислительные системы, которые находят применение

при решении научно-технических задач, использующих векторные и матричные структуры данных.

В векторной вычислительной системе обязательно имеется векторный процессор (рис. 17.9), содержащий векторные регистры для данных в формате с плавающей запятой и векторное арифметико-логическое устройство, обрабатывающее данные в этом формате. Векторные регистры и векторное арифметико-логическое устройство, по сути дела, представляют собой группу обычных регистров и обычных арифметико-логических устройств. Количество устройств в группе определяет максимальную размерность векторов, над которыми векторный процессор может выполнять действия. В силу ограничений по стоимости системы это количество обычно не превышает 64.

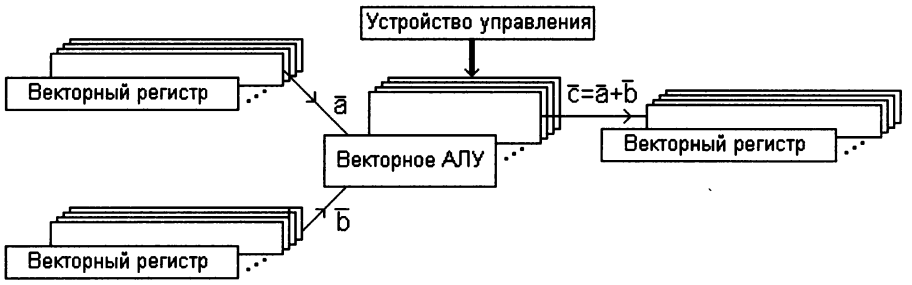


Рис. 17.9. Упрощенная схема векторного процессора систем класса ОКМДС

Векторные регистры содержат компоненты векторных операндов. По команде из единого устройства управления они поступают в векторное арифметико-логическое устройство, которое обеспечивает выполнение одной и той же заданной операции над всеми компонентами векторов. Вектор-результат поступает в векторный регистр.

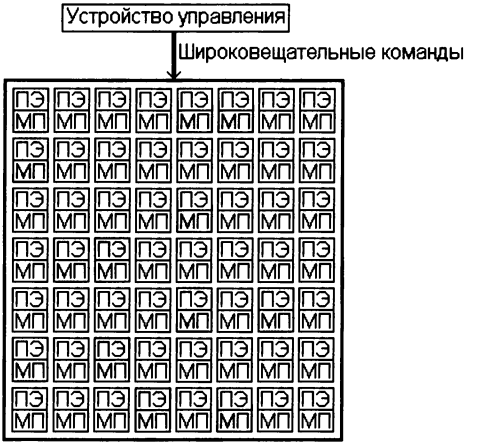
Векторные процессоры, изображенные на рис. 17.9, не получили широкого распространения из-за высокой стоимости организации векторного арифметико-логического устройства. Более приемлемыми по стоимости оказались **векторно-конвейерные** вычислительные системы, в которых векторные регистры комбинируются с конвейерным арифметико-логическим устройством. В таких системах размерности векторов были существенно увеличены. Выдача устройством управления единственной команды, которая должна быть выполнена над группой данных, и их бесперебойная подача на конвейер из векторных регистров обеспечивают постоянную загрузку конвейера и его высокую производительность. Типичным представителем векторно-конвейерных систем является суперкомпьютер Cray-1, разработанный под руководством Сеймура Крея в компании Cray Research (впоследствии — подразделение фирмы Silicon Graphics). В то время (1970-е гг.) это была самая мощная вычислительная система в мире. Ее тактовая частота составляла 80 МГц, объем оперативной памяти был равен 8 Мбайт, а пиковая производительность составляла 160 Мфлоп.

Примером более современных векторно-конвейерных вычислительных систем является разработанный в начале 1990-х годов в той же фирме суперкомпьютер

Cray T90. Эта система может содержать до 16 процессорных элементов, использующих общую оперативную память. Тактовая частота процессоров равна 250 МГц, а пиковая производительность достигает 32 Гфлоп. Можно упомянуть еще и использующую векторные процессоры вычислительную систему Earth Simulator компании NEC. Система была пущена в эксплуатацию в 2002 г. В это время она состояла из 640 вычислительных узлов, в каждом из которых находилось по 8 процессоров. Пиковая производительность всей системы из 5120 процессоров составляла 40 Тфлоп.

В матричных процессорах, которые известны также под названием **массивно-параллельных** процессоров, каждый процессорный элемент имеет выделенный локальный блок оперативной памяти. Совокупность связанных с блоками памяти процессорных элементов образует матричную конфигурацию. Единое устройство управления выдает всем процессорным элементам одну и ту же команду, которая синхронно выполняется над различными потоками данных, поступающими в процессорные элементы независимым образом из собственных модулей памяти.

Типичным представителем матричных систем является суперкомпьютер ILLIAC IV, упрощенная схема которого изображена на рис. 17.10. Вычислительный узел этой системы содержал процессорный элемент и модуль памяти из 2048 слов по 64 бита, что составляет 16 Кбайт в современном исчислении. Каждый процессорный элемент имел доступ только к собственной памяти, но при необходимости он мог связаться с любым из четырех своих непосредственных соседей. Единое устройство управления системы одновременно выдавало всем процессорным элементам одну и ту же команду, и каждый элемент выполнял ее над собственными операндами, находящимися в его локальной памяти. Попутно заметим, что режим передачи информации, при котором сообщение посылает один узел, а принимают сразу все остальные узлы, называется **широковещательным**.



Матрица процессорных элементов(ПЭ)/модулей памяти (МП) 8×8 узлов

Рис. 17.10. Упрощенная схема вычислительной системы ILLIAK IV

Сначала планировалось создать систему, включающую матрицу из 16 × 16 вычислительных узлов. Хотя система строилась с 1967 по 1975 г., ее действующий вариант содержал только один квадрант планируемой матрицы размерностью 8 × 8 узлов с пиковой производительностью 10 Гфлоп. Суперкомпьютер ILLIAC 4 находился в эксплуатации до 1982 г.

Другой разновидностью систем класса ОКМД являются системы ОКМДР с поразрядной обработкой данных. Единственными представителями этого класса считаются **ассоциативные** системы, в которых процессор выбирает данные не по адресу, а по значению (см. 9.1.3). Системы с ассоциативными процессорами оперируют не совокупностью разрядов одного и того же слова, а совокупностью разрядов с одним и тем же номером из различных слов оперативной памяти. Такую совокупность называют **разрядным срезом**.

На рис. 17.11 изображена схема, иллюстрирующая разницу между адресной выборкой в обычных процессорах и выборкой разрядного среза в ассоциативных процессорах. В примере рассматривается блок оперативной памяти, состоящий из 32 четырехбайтовых полей. Во время выполнения адресной выборки (рис. 17.11, *а*) в процессорный элемент передается 32-битовое поле с заданным адресом. В примере на рисунке выбирается поле с адресом $00AAFE0C_{16}$. Во время ассоциативной выборки разрядного среза (рис. 17.11, *б*) в процессорный элемент передается также группа из 32 битов, но они выбираются из одного и того же разряда всех полей. В примере на рисунке биты выбираются из 20-го разряда всех 32 рассматриваемых полей.

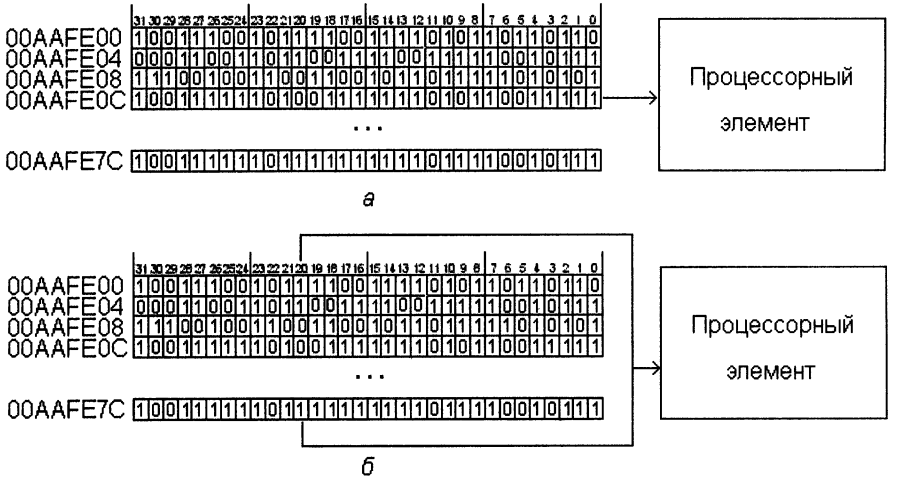


Рис. 17.11. Выборка из оперативной памяти: *а* — адресная; *б* — ассоциативная

При наличии нескольких процессорных элементов различные разрядные срезы поступают на обработку в разные элементы и обрабатываются одновременно. Это позволяет выделить слова памяти с желательным содержимым в тех или иных

разрядах. Такой подход находит применение в специализированных вычислительных системах, таких как, например, **машины баз данных**, которые обеспечивают быстрое выполнение операций поиска в базах данных.

Архитектура **МКОД**, или **MISD** (от Multiply Instruction Single Data stream — множественный поток команд обрабатывает одиночный поток данных) (рис. 17.12, *а*). Подразумевается наличие нескольких процессорных элементов, которые одновременно обрабатывают один и тот же поток данных, выполняя над ним одни и те же или разные операции. Некоторые специалисты относят к этому классу конвейерные системы. Однако в конвейере отсутствует *одновременность* выполнения действий ступенями конвейера над одним и тем же кодом. Разные ступени обрабатывают один код последовательно. Одновременно разные ступени конвейера работают над разными кодами данных. По мнению других специалистов, класс МКОД пустой. Мы также будем считать его пустым, а конвейерные процессоры отнесем к классу ОКОД для скалярных аргументов или к классу ОКМД в случае обработки векторных данных (векторно-конвейерные системы).

Рис. 17.12. Архитектуры с множественным потоком команд: *а* — МКОД; *б* — МКМД

Архитектура **МКМД** или **MIMD** (от Multiply Instruction Multiply Data stream — множественный поток команд, обрабатывающих множественный поток данных). Бытовым аналогом работы таких систем является ситуация, в которой несколько человек одновременно делают несколько различных деталей одного и того же устройства, выполняя одинаковые или различные операции над ними.

В системах МКМД имеется группа процессорных элементов, каждый из которых работает под управлением отдельного устройства управления и обрабатывает локальные или общие данные (рис. 17.12, *б*). Это самый многочисленный класс параллельных систем. К нему относятся, например, уже упоминавшиеся системы Intel Paragon, «Эльбрус» и многие другие вычислительные системы.

Классификация Флинна в классе МКМД выделяет пустой подкласс МКМДР с поразрядной обработкой и подкласс МКМДС, содержащий слабосвязанные системы типа МКМДС/Нс и сильносвязанные системы типа МКМДС/Вс. Напомним, что системы со слабой связью называются многомашинными (или мультикомпьютерами), а системы с сильной связью — многопроцессорными (или мультипроцессорами).

17.4. Классификация параллельных систем класса МКМД

Классификация Флинна была предложена в 1966 г. К настоящему времени она хорошо описывает особенности вычислительных систем только на нескольких верхних уровнях, изображенных на рис. 17.7. Более детальная классификация, в частности, классификация систем типа МКМД, уже не соответствует современным особенностям класса.

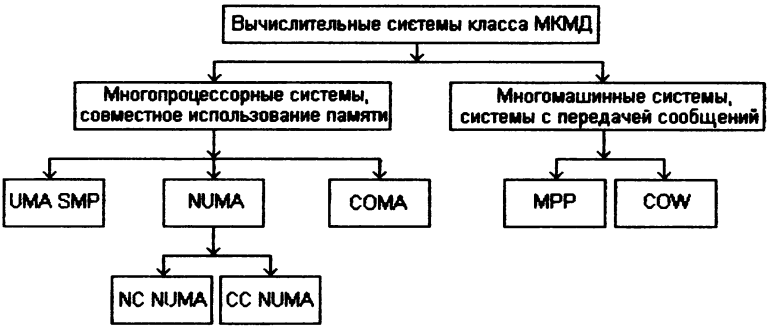


Рис. 17.13. Современная классификация систем класса МКМД

На рис. 17.13 приведена уточненная структура вычислительных систем класса МКМД. По данной классификации вычислительные системы этого типа делятся на многопроцессорные системы, совместно использующие оперативную память, и многомашинные системы, осуществляющие обмен с помощью передачи сообщений. Каждый из классов имеет определенные достоинства и недостатки. Преимуществами многопроцессорных систем с общей памятью являются:

- совместимость с хорошо отлаженными механизмами доступа к оперативной памяти, используемыми в однопроцессорных системах;
- простота программирования приложений, простота разработки трансляторов и операционных систем;
- малое время доступа, более высокая производительность линий связи;
- возможность использования аппаратно управляемого кэширования.

К преимуществам систем с передачей сообщений относятся:

- более простое аппаратное устройство, не требующее согласованности кэш-памяти различных процессорных элементов;
- возможность построения хорошо масштабируемых систем.

В класс систем с общей памятью входят две основных группы, UMA и NUMA. Вычислительные системы класса UMA (от Uniform Memory Access — унифицированный, однородный доступ к памяти) обеспечивают всем процессорным элементам системы однородный доступ к общему адресному пространству

оперативной памяти. Любой из процессорных элементов может записать в любое поле памяти какой угодно код, после чего любой другой процессорный элемент может использовать этот код любым образом. Каждый процессорный элемент имеет одно и то же время доступа к любому полю оперативной памяти. Поскольку все процессорные элементы находятся в одних и тех условиях и имеют одинаковые права доступа к памяти, такие системы называют также **симметричными мультипроцессорными системами** и обозначают **SMP** (от Symmetric Multi-Processing).

Связи между процессорными элементами и модулями оперативной памяти организуются по высокоскоростным общим шинам и/или с помощью матричных коммутаторов. Количество процессорных элементов в системе класса SMP обычно не превышает 32–64. В такой системе каждый процессорный элемент выполняет свою программу практически не зависящим от остальных процессоров образом. Все процессорные элементы обычно работают под управлением единственной общей для всех процессоров операционной системы.

Каждый процессорный элемент системы может иметь собственный кэш. Эта возможность вызывает проблему когерентности кэшей различных процессорных элементов, которая в целом аналогична проблеме когерентности кэша и оперативной памяти. Чтобы обеспечить согласованность данных в кэшах разных процессоров, доступ процессорных элементов к оперативной памяти реализуется на базе различных протоколов, обычно использующих принцип взаимоисключающего доступа.

Системы типа SMP просты в эксплуатации, не слишком дороги, но при этом отличаются относительно невысокой масштабируемостью. Характерным примером системы этого типа является компьютер Sun Enterprise 10000, состоящий из 64 процессорных элементов Ultra SPARC III.

Системы NUMA (от Non Uniform Memory Access) — с неоднородным доступом к памяти — состоят из нескольких однородных базовых модулей, которые включают несколько процессорных элементов со своими локальными блоками оперативной памяти, а также блока общей для всех процессоров оперативной памяти. При этом вся оперативная память физически распределена между процессорными элементами, но логически является общей, с единым адресным пространством. По причине физического разделения доступ процессорного элемента к собственной локальной памяти оказывается в несколько раз быстрее, чем доступ к общему блоку памяти и к блокам локальной памяти других процессорных элементов. Время доступа к локальной и общей памяти в системах NUMA может различаться в 5–10 раз. Эти системы имеют очень хорошие возможности масштабирования, количество процессорных элементов в них может доходить до нескольких тысяч.

В архитектуре NUMA также существуют проблемы, связанные с доступом процессорных элементов к данным, модифицированным другим процессорным элементом и помещенным в его кэш-память. В связи с этим в класс NUMA входят системы без кэширования NC-NUMA (от No Caching NUMA) и системы с согласованной кэш-памятью CC-NUMA (от Coherent Cache NUMA), в которых проблемы когерентности решаются довольно эффективно. Характерным примером вычис-

лительной системы класса NC-NUMA является машина Carnegie-Mellon Cm*. Системами CC-NUMA являются: суперкомпьютеры Hewlett-Packard Superdome, SGI Origin 3000, Sun HPC 15000 и Sequent NUMA-Q 2000.

Вообще говоря, к группе многопроцессорных систем относятся еще и системы класса **COMA** (от Cache Only Memory Access), в которых локальная память каждого процессорного элемента используется только как кэш-память. При этом общая для всех процессорных элементов оперативная память в системе отсутствует. Отличительной особенностью этой архитектуры является отсутствие постоянной привязки строк кэша к адресам локальных блоков оперативной памяти. Строки помещаются в локальную память (кэш) любого из процессорных элементов по мере необходимости. Это избавляет от проблем согласования, но появляются сложности с удалением элементов, а также с определением наличия строк кэша в оперативной памяти. Эти системы пока не очень распространены, и еще не накоплен достаточный опыт их эксплуатации.

В классе многомашинных систем выделяют две группы: **массивно-параллельные системы MPP** (от Massively Parallel Processor) и **кластерные системы COW** (от Cluster Of Workstation). Системы класса MPP называют также системами **с массовым параллелизмом**, а кластерные системы иногда обозначают как NOW (от Network Of WorkStation).

ПРИМЕЧАНИЕ

Не следует путать матричные или массивно-параллельные процессоры, относящиеся к классу ОКМД, с массивно-параллельными системами MPP, относящимися к классу МКМД. В первом случае речь идет об одном компьютере, который содержит много процессоров. Во втором — о взаимосвязанных компьютерах, которые могут одновременно решать одну и ту же задачу. Во избежание путаницы в первом случае рекомендуется использовать термин «матричные процессоры».

Массивно-параллельные системы MPP состоят из однородных вычислительных узлов, включающих один или несколько процессорных элементов, локальную память для каждого элемента, модули ввода/вывода и коммуникационный узел или сетевой адаптер, которые связаны специализированными высокоскоростными линиями связи. Непосредственный доступ к модулю памяти имеет только его процессорный элемент. Можно заметить, что оперативная память физически разделена между процессорными элементами, так же как и в машине ILLIAC IV класса ОКМД.

В системах типа MPP полноценная операционная система, как правило, работает только на одной управляющей машине, а на каждом из процессорных элементов функционирует ее урезанный вариант. Хотя имеются и варианты, когда на каждом процессоре работает полноценная операционная система типа Unix.

Фактически это системы с распределенными между несколькими компьютерами аппаратными и программными ресурсами. Причем эти компьютеры находятся на относительно небольшом удалении друг от друга и связаны между собой высокоскоростной сетью линий связи, по которым происходит обмен сообщениями.

Системы класса MPP отличаются высокой, практически стопроцентной отказоустойчивостью и очень хорошо масштабируются. Например, вычислительная система ASCI White (от Accelerated Strategic Computing Initiative – ускоренная стратегическая вычислительная инициатива) размером в два баскетбольных поля содержит 8192 процессора. Известны системы, содержащие до 65 536 процессоров.

Системы COW, или кластерные архитектуры, представляют собой объединение нескольких стандартных персональных компьютеров и/или серверов посредством стандартных сетевых средств связи. Машины, входящие в кластерную систему, могут использоваться для совместного и одновременного выполнения одной и той же программы.

ВНИМАНИЕ

Вычислительным кластером называется совокупность компьютеров, объединенных в рамках локальной или глобальной сети для решения одной задачи. В качестве вычислительных узлов используются однопроцессорные персональные компьютеры, 2- или 4-процессорные системы.

Кластерные архитектуры фактически представляют собой более дешевый вариант массивно-параллельных систем, вариант, построенный из стандартных персональных машин и стандартных средств сопряжения в локальную или глобальную сеть.

Каждый узел кластера работает под управлением своей собственной операционной системы (Linux, Windows 2000, Solaris). Кластер отличают развитая коммуникационная среда и наличие удаленного доступа к нему через Интернет.

Один из первых кластерных проектов был реализован в 1994 г. Он был создан из 16 процессоров Intel 80486 с тактовой частотой 100 МГц. В каждом узле кластера размещались модуль оперативной памяти объемом 16 Мбайт и три сетевых адаптера Ethernet.

Некоторые специалисты выделяют в группе многомашинных систем еще и класс **распределенных** вычислительных систем, подразумевая наличие в составе системы большого количества пространственно рассредоточенных аппаратных и программных ресурсов, функционирующих автономно, но согласованно под управлением единой операционной системы верхнего уровня и/или собственных, локальных операционных систем. Машины, входящие в распределенные системы, не имеют ни общих блоков оперативной памяти, ни общих периферийных устройств. Распределенные системы отличаются от кластеров тем, что компьютеры, входящие в такую систему, могут принадлежать разным собственникам и находиться на значительном удалении друг от друга. Такие системы используются для распределенного хранения и обработки информации.

ВНИМАНИЕ

Хранение и не синхронизированная во времени обработка связной информации, осуществляемые на пространственно разделенных вычислительных машинах, называются распределенными.

Необходимо различать параллельную обработку в централизованных и сетевых системах и распределенную обработку в сетевых и распределенных системах. Под параллельной обычно понимают обработку, синхронизированную во времени, в то время как обработка, осуществляемая асинхронно, считается распределенной.

Частным случаем распределенных вычислительных систем считаются **вычислительные сети**, **метакомпьютеры** или **Grid**-системы (от grid — решетка). Они отличаются от кластеров большей независимостью и сложностью образующих систему компонентов. В вычислительных сетях обычно отсутствуют операционные системы верхнего уровня. Обычно Grid-системами считаются серверы и системы хранения данных, объединенные в единую группу вычислительных ресурсов. Отличительной чертой Grid-системы является согласованная работа входящих в систему компьютеров над одной и той же задачей.

Метакомпьютер обладает огромными вычислительными распределенными неоднородными ресурсами, принадлежащими разным организациям. Мощность метакомпьютера значительно превосходит мощность любого современного суперкомпьютера. Например, с помощью метакомпьютера решена задача вычисления максимального для сегодняшних возможностей числа Мерсенна вида $2p - 1$, где p — простое число. В решавший эту задачу метакомпьютер вошли свыше двадцати тысяч связанных по Интернету персональных компьютеров по всему миру. Вычислительная сеть работала над этой задачей два с половиной года, в результате в ноябре 2001 г. было найдено значение числа $2^{13466917} - 1$.

Другим важнейшим частным случаем распределенных вычислительных систем являются **компьютерные сети**, в которых независимо используются компьютеры, объединенные линиями связи. В связи с огромной важностью компьютерных сетей более детальное их обсуждение вынесено в часть III учебника. А более подробный общий обзор параллельных вычислительных систем можно найти в [6], [8], [11], [28], [30]

Контрольные вопросы и упражнения

1. Охарактеризуйте ограничения, присущие однопроцессорным вычислительным системам.
2. Поясните смысл терминов «параллельная программа», «многопроцессорная система», «многомашинная система».
3. Что называется ускорением параллельной вычислительной системы?
4. Сформулируйте законы Амдала. Какой практический вывод из них следует?
5. Перечислите базовые характеристики параллельных вычислительных систем.
6. Что понимается под топологией системы? Какую роль играет топология? Какие существуют топологии?
7. Какую роль играют коммутаторы в архитектуре параллельных систем?
8. Сравните возможности основных типов коммутаторов.

9. Охарактеризуйте базовые топологии параллельных систем.
10. Поясните смысл терминов «поток команд», «множественный поток команд», «поток данных», «множественный поток данных», «косвенная слабая связь», «прямая слабая связь», «сильная связь».
11. Перечислите классификационные признаки классификации параллельных систем по Флинну.
12. Изобразите дерево классификации параллельных систем по Флинну.
13. Охарактеризуйте класс ОКОД.
14. Охарактеризуйте класс ОКМД. Какие типы вычислительных систем входят в этот класс? Приведите примеры.
15. Охарактеризуйте отличительные особенности векторных, векторно-конвейерных и матричных систем.
16. Опишите особенности ассоциативных систем и укажите основные области их применения.
17. Охарактеризуйте класс МКОД. Какие системы относятся к этому классу?
18. Охарактеризуйте класс МКМД. Изобразите дерево классификации класса.
19. Охарактеризуйте преимущества систем с общей памятью и систем с передачей сообщений.
20. Дайте сравнительную характеристику классов UMA и NUMA.
21. Чем отличаются друг от друга классы CC-NUMA и NC-NUMA.
22. Дайте сравнительную характеристику классов MPP и COW.
23. Поясните смысл терминов «вычислительный кластер», «распределенная система», «вычислительная сеть», «метакомпьютер», «Grid-система», «компьютерная сеть».

Глава 18
Неклассические архитектуры

В группу вычислительных систем с неклассической архитектурой можно отнести специализированные системы, системы, управляемые потоком данных, и системы, работающие на физических принципах, отличающихся от используемых в электронных компьютерах.

К специализированным системам относятся прежде всего **машины баз данных**, **машины языков высокого уровня** и всевозможные управляющие системы, которые обеспечивают работу самых разных объектов, начиная от мобильного телефона, стиральной машины и заканчивая космическими аппаратами и оборонными комплексами. Базы данных и их приложения в современном мире являются одной из важнейших сфер применения информационных технологий. Системы управления базами данных, собственно базы данных и их приложения обычно реализуются на универсальных компьютерах с традиционной архитектурой, а также на мощных вычислительных комплексах параллельного действия. Вместе с тем такая реализация баз данных приводит к большим накладным расходам, неудовлетворительной скорости и недостаточно высокой надежности работы с ними. Специалисты нашли выход в создании специализированных вычислительных систем, архитектура которых приспособлена к специфическим задачам и действиям, характерным для баз данных и их приложений. Как правило, в этих системах используются различные реализации ассоциативной памяти и ассоциативных процессоров. Характерным примером машин баз данных считается машина DBC (от database computer), построенная в университете штата Огайо (США).

В архитектурах машин языков высокого уровня организация кодов данных и машинных команд привязывается к структурам данных и операциям над ними, характерным для того или иного языка программирования, например языка Кобол, PL/1, Паскаль, Ада и т. д. В классических архитектурах единственной структурой данных, которая для обеспечения максимально высокой скорости работы привязана к возможностям аппаратных средств, является последовательность

битов, занимающих некоторое поле памяти. Наделение этой последовательности свойствами, превращающими ее в структуры данных, используемые в программах и необходимые для решения реальных задач, возлагается на программное обеспечение, на трансляторы, инструментальные среды и т. д. Это значительно усложняет трансляторы и исполнительные системы. В архитектурах машин языков высокого уровня необходимые для языка структуры данных и операции над ними реализуются на аппаратном уровне, что влечет за собой их усложнение. Такой подход позволяет либо существенно упростить процесс трансляции на машинный уровень, либо вообще отказаться от него. Разработчиками было создано довольно много таких машин. Можно упомянуть, например, системы на процессорах фирмы Western Digital, ориентированные на язык Паскаль, машину iAPX 432 фирмы Intel, в некоторой степени реализующую возможности языка Ада, и т. д. Подробное обсуждение архитектур, связанных с машинами баз данных и языками высокого уровня, можно найти, например, в [24].

К группе неклассических архитектур, в которых последовательность действий определяется потоком обрабатываемых данных, относятся потоковые и систолические компьютеры, а также нейрокомпьютеры.

Характерная особенность **потоковых компьютеров**, или компьютеров, управляемых данными **DFC** (от Data Flow Computers), — принципиальное отсутствие регистра счетчика команд, который определяет порядок выполнения действий, то есть поток команд. В таких машинах последовательность выполнения операций определяется структурой обрабатываемых данных, отображенной на граф потока данных. Потоковая обработка данных осуществляется в соответствии с принципом: всякое действие выполняется в тот момент, когда для него готовы данные. Ясно, что вычислительные системы, точно выполняющие указанный принцип, обеспечивают максимально возможное для данного алгоритма распараллеливание.

Чтобы более полно представить способ работы потоковых компьютеров, рассмотрим пример вычисления значения выражения $P = (a + b)d + (a + d)b + dbc$. Анализ входящих в это выражение слагаемых показывает, что вычисления значений выражений в скобках $(a + b)$, $(a + d)$ и произведения db могут производиться одновременно. Поручив выполнение этих действий трем процессорным элементам, можно за время выполнения одного действия получить значения трех промежуточных величин. На втором шаге, также используя три процессорных элемента, можно одновременно получить значения всех трех слагаемых. Дальнейшие действия по вычислению значения P могут быть выполнены только последовательно. В строках табл. 18.1 представлены действия, которые могут быть выполнены параллельно, а на рис. 18.1 — граф потока данных для данного примера.

Таблица 18.1. Распараллеливание действия для вычисления значения P

1-е действие	2-е действие	3-е действие	4-е действие
$r1 := a + b$	$r1 := r1 \times d$	$r1 := r1 + r2$	$P := r1 + r3$
$r2 := a + d$	$r2 := r2 \times b$		
$r3 := d \times b$	$r3 := r3 \times c$		

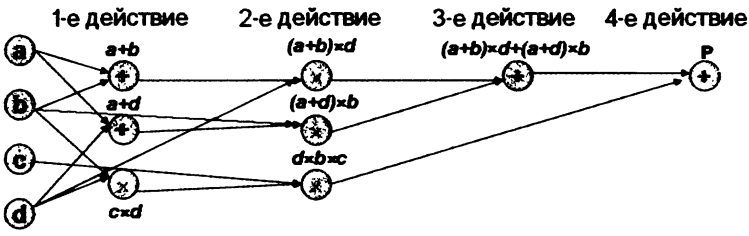


Рис. 18.1. Граф потока данных

Таким образом, имея в своем распоряжении три процессорных элемента и организовав выполнение действий в порядке, изображенном на графе, можно сократить количество этапов вычисления с 8 до 4, то есть уменьшить время вычислений примерно вдвое. В принципе, можно предложить другой граф и другую последовательность действий, которые дают тот же самый результат, но увеличить количество параллельно выполняемых действий и уменьшить общее количество этапов для данного примера не удастся.

В общем случае для произвольного алгоритма универсальная потоковая машина должна содержать неограниченно большое количество процессорных элементов. Чтобы иметь возможность реализовать любой потоковый граф, эти элементы должны находиться в узлах системы линий связи с полносвязной топологией. Очевидно, что реализация универсальной потоковой машины наталкивается на непреодолимые сложности. Очевидно также, что в реальных потоковых машинах из-за ограниченного количества процессорных элементов максимально высокий уровень параллелизма не достигается. В связи с этим реальные потоковые машины используются не как универсальные, а как специализированные, в которых граф потока данных либо заранее известен и фиксирован, либо изменяется не очень значительно. В этом случае на алгоритмах из некоторого фиксированного класса возможно достижение максимально высокого параллелизма и, следовательно, максимального выигрыша по сравнению с однопроцессорными компьютерами и даже по сравнению с классическими универсальными многопроцессорными системами.

Разновидностью потоковых машин являются системы, которые называют **систолическими массивами** (или машинами). Это специализированные системы, которые удобно использовать для реализации алгоритмов, построенных на итерационных численных методах решения математических задач. Рассмотрим, например, типичную итерационную схему вида

$$x_i^0 = a_i;\ x_i^j = f_i(x_1^{j-1}, \ldots, x_n^{j-1}),\ i = 1, 2, \ldots, n;\ j = 1, 2\ldots$$

где $\bar{x} = \{x_1, x_2, \ldots, x_n\}$ — искомый вектор, $\bar{a} = \{a_1, a_2, \ldots, a_n\}$ — вектор начальных приближений, $\bar{f} = \{f_1, f_2, \ldots, f_n\}$ — вектор-функция, задающая связь между вычисляемыми на текущем (j-м) шаге значениями компонент вектора $\bar{x}$ и их значениями на предыдущем шаге.

Вычисления в систолической системе выполняются по схеме, показанной на рис. 18.2. В начальный момент в систему вводятся начальные значения a_i,

i =1, 2, ..., n, а затем в параллельном режиме, так же как в потоковых машинах, производится вычисление значений $f_i(x_1^{j-1}, ..., x_n^{j-1})$, которые определяют компоненты x_i на следующей итерации. Эти значения затем вновь поступают на вход системы. После достижения нужной точности вычисления завершаются.

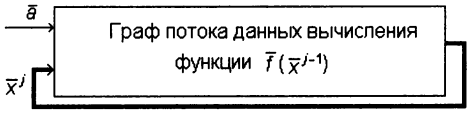


Рис. 18.2. Упрощенная схема работы систолической машины

Обсуждаемая система осуществляет вычисления как бы в «пульсирующем» режиме, отсюда и ее название — систолическая (греческое слово «систола» означает сжатие, сокращение, ритмические сокращения сердца).

Нейрокомпьютеры — это специализированные компьютеры, действие которых основано на теории так называемых персептронов (от perception — восприятие, представление), то есть устройств, моделирующих процесс восприятия внешней среды при помощи сети нейронов.

Напомним, что нейроном называется нервная клетка биологических организмов. Нейроны через аксоны (отростки) имеют многочисленные входные и выходные связи с аналогичными нервными клетками организма (рис. 18.3, *а*). Если импульсы, поступившие в нейрон по входным линиям связи, превышают некоторое пороговое значение, то нейрон возбуждается и переходит в активное состояние, передавая на выходные связи импульс возбуждения. Оказывается, что такая структура обеспечивает хорошую обучаемость всей системы, а стало быть, и всего биологического организма в целом.

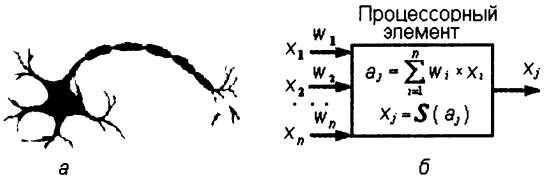


Рис. 18.3. Нейрон: *а* — увеличенное изображение; *б* — функциональная модель

Нейрокомпьютер представляет собой сеть очень простых процессорных элементов, которые так же, как и нейроны, объединены между собой линиями связи. Топология линий связи может представлять однослойные или многослойные структуры. В каждый момент времени все процессорные элементы — «нейроны» — работают параллельно. На вход каждого процессорного элемента от других связанных с ним процессорных элементов поступает некоторое входное воздействие x_i, i = 1, ..., n (рис. 18.3, *б*). Все процессорные элементы одновременно вычисляют суммарные воздействия

$$a_j = \sum_{i=1}^{n} w_i x_i, j = 1, 2, ..., n,$$

в которых входные воздействия x_i учитываются с некоторыми весами w_i. Далее суммарное воздействие a_j на j-процессорный элемент сравнивается с пороговым, и при необходимости этот процессорный элемент переводится в активное состояние с выдачей соответствующего выходного сигнала $x_j = S(a_j)$, где S — функция активации процессорных элементов.

В начальный момент времени система процессорных элементов подготавливается, настраивается с помощью задания пороговых значений для каждого элемента и весовых коэффициентов линий связи. На вход системы подается определенная начальная совокупность сигналов, и в системе осуществляется дискретное в параллельном режиме вычисление функций активации для каждого процессорного элемента. Считается, что сеть пришла в равновесное состояние, когда перестают изменяться значения выходных сигналов процессорных элементов. Можно считать, что в этом случае некоторый итерационный процесс сошелся.

Распределяя определенным образом веса связей и пороговые значения для процессорных элементов, можно настраивать сеть на решение задач различных классов. Нейрокомпьютеры можно «обучать», изменяя по определенному алгоритму параметры сети процессорных элементов таким образом, чтобы на заданных образцах система давала правильный ответ, а затем предъявлять для «опознания» не рассматривавшийся ранее объект. Для нейрокомпьютеров отсутствуют стандартные понятия программирования, которые заменяются легко автоматизируемым процессом обучения. Обученный нейрокомпьютер по эффективности значительно превосходит традиционные компьютеры. Основными областями применения нейрокомпьютеров в настоящее время являются: распознавание образов, задачи искусственного интеллекта, поиск точек экстремума, оптимизационные задачи.

Нейронные компьютеры отличаются крайней простотой процессорных элементов, высокой степенью параллелизма, по сути дела, распределенным характером вычислений, надежностью функционирования, высокой отказоустойчивостью, отсутствием традиционного программирования, возможностью решения слабо формализованных и неформальных задач.

В настоящее время многие специалисты прогнозируют полное исчерпание ресурсов дальнейшего развития и наращивания эффективности вычислительных систем, созданных по кремниевым технологиям, к 2010–2015 гг. В связи с этим ведутся многочисленные исследования по использованию в вычислительных системах физических принципов, отличных от используемых в полупроводниковых машинах. В частности, можно упомянуть разработки **квантовых**, **оптических**, **биологических** и некоторых других видов компьютеров.

В квантовых компьютерах основным логическим элементом является отдельный атом вещества. Такой элемент называется **кубитом** (от qubit — quantum bit — квантовый бит). Квантовые свойства атомов позволяют организовать естественный параллелизм хранения информации и вычислений, так как кубит в один и тот же момент времени может содержать и единицу, и нуль. Кроме того, размеры устройств и плотность хранения информации в этом случае изменятся в сотни тысяч и миллионы раз.

В оптических компьютерах оптоэлектронные логические схемы и запоминающие устройства, теоретически, обеспечивают улучшение рабочих характеристик в десятки миллионов раз, так как частоты оптических устройств в сотни миллионов раз выше частот, используемых в современных электронных системах.

В машинах, основанных на биотехнологических подходах, центральное место занимают так называемые ДНК-процессоры, которыми являются молекулы ДНК (от дезоксирибонуклеиновой кислоты), играющие важную роль в передаче наследственных признаков у биологических организмов. ДНК-память представляет собой цепочку из четырех нуклеотидов, являющихся базовыми блоками молекул ДНК. Операциям над данными в ДНК-процессоре соответствуют биохимические операции с молекулами ДНК.

Перспективные возможности таких вычислительных систем весьма велики: например, предполагается, что за счет одновременного вступления в реакцию нескольких триллионов молекул скорость работы такой параллельной системы может составить до 10^{14} операций в секунду при весьма простой структуре и незначительных размерах. Плотность памяти за счет того, что используются не бинарные устройства памяти, а четверки нуклеотидов, должна возрасти в триллионы раз. Вместе с тем на современном уровне развития биотехнологий существует масса нерешенных проблем, в частности, проблем со считыванием результатов работы ДНК-процессора, невозможностью длительного хранения результатов из-за самораспада молекул ДНК и т. д.

Дополнительный материал по обсуждавшимся в главе вопросам можно найти в изданиях [6], [8], [11], [12], [19], [24], [27], [30], [32].

Контрольные вопросы и упражнения

1. Сравните архитектуры машин баз данных и машин языков высокого уровня.
2. Сформулируйте принципиальные различия систем, управляемых потоком команд, и систем, управляемых потоком данных.
3. Какой принцип положен в основу формирования графа потока данных?
4. Опишите особенности систолических машин.
5. Опишите устройство и работу нейрокомпьютеров.
6. Опишите перспективные направления разработок вычислительных систем.

Часть III

Введение в архитектуру компьютерных сетей

В начальный период развития вычислительной техники пользователю и его программам предоставлялся непосредственный доступ ко всем ресурсам компьютера, которые поступали в монопольное владение единственной выполняющейся программы. В тот период операционных систем еще не было, поэтому много времени занимал переход от выполнения одной программы к выполнению другой. Такой переход осуществлялся пользователем с помощью ручного управления компьютером. Человеку приходилось самостоятельно выполнять последовательность мелких управляющих операций, которые в настоящее время автоматизированы и выполняются операционными системами.

Для уменьшения непроизводительных потерь на следующем этапе развития был внедрен режим **пакетной обработки**. Пользователь, вообще не имевший доступа в машинный зал, готовил программу и данные на перфокартах. Набор перфокарт представлял собой отдельное задание вычислительной системе. Из заданий различных пользователей формировался пакет, который передавался вычислительной системе ее оператором в удобное время. Операционная система автоматически выбирала из пакета очередное задание, загружала его в память и после окончания выполнения обеспечивала вывод результатов на печать и переход к следующему заданию. Пользователь получал результаты выполнения программы, как правило, на следующий день.

Большой временной разрыв между передачей программы на выполнение и получением результатов порождал массу неудобств для программистов. Чтобы повысить производительность их работы, к машинам стали подключать несколько терминалов, которые находились в специально выделенном помещении или на рабочих местах специалистов. Так появился **многотерминальный** режим. Работа программистов с вычислительной системой осуществлялась в режиме **разделения времени**.

Это значит, что все выполняющиеся программы имеют равные права по отношению к ресурсам компьютера. Центральному процессору каждая из программ передается на определенный отрезок времени, довольно малый с точки зрения человека, но достаточный для того, чтобы процессор выполнил значительную последовательность команд программы. В результате каждый пользователь получает иллюзию единоличной, хотя и немного замедленной работы с вычислительной системой.

Для получения доступа к мощным машинам, возможно, удаленным на тысячи километров, стали использовать терминалы, подсоединяемые к компьютерам с помощью модемов и телефонных сетей или по специально прокладываемым кабельным линиям. Многотерминальный удаленный режим подготовил почву для соединения друг с другом компьютеров. Такое соединение по аналогии с телефонным стали называть **сетью ЭВМ**, **компьютерной сетью**, **информационной сетью**, **сетью передачи данных** или просто сетью.

ВНИМАНИЕ

Компьютерной сетью называется частный случай распределенных вычислительных систем, объединение двух и более компьютеров линиями связи, с помощью которых можно выполнять обмен данными и программами между любыми включенными в сеть компьютерами, а также осуществлять совместную или независимую обработку данных.

В сетевых объединениях компьютеров появились специальные службы электронной почты, обмена файлами, доступа к базам данных и т. д. Так возникли первые **глобальные** сети.

Из-за высокой стоимости компьютеров на начальных этапах развития одно предприятие, как правило, не могло приобрести и соединить между собой несколько компьютеров. Позднее, когда компьютеры стали намного дешевле и предприятия и организации получили возможность устанавливать у себя более одного компьютера, по образцу глобальных стали создавать **локальные** сети, которые в масштабах одного предприятия обеспечивали пользователей практически теми же самыми возможностями, что и глобальные сети.

Компьютерные сети являются особой разновидностью многомашинных систем, в которых связь между полноценными компьютерами осуществляется через сетевые адаптеры или модемы и пространственно протяженные линии связи. Каждый подключенный к сети компьютер работает под управлением имеющей специальные сетевые функции собственной операционной системы. Общая операционная система обычно отсутствует. Взаимодействие между машинами происходит при помощи сообщений, передаваемых по линиям связи. С помощью сообщений передаются данные и запрашиваются необходимые ресурсы.

К числу достоинств компьютерных сетей относятся:

- возможность оперативного, практически мгновенного обмена информацией между любыми пользователями, имеющими доступ к компьютерам сети;
- возможность совместного использования дорогостоящей и эффективной аппаратуры, включенной в состав сети (например, лазерных принтеров);

- возможность совместного использования программ и данных, хранящихся в компьютерах сети, что позволяет экономить дисковую память из-за отказа от дублирования файлов на каждом из компьютеров;
- доступ к уникальной, то есть имеющейся в единичных экземплярах, информации для большого числа людей;
- возможность использования для обработки информации более мощных компьютеров;
- возможность объединения вычислительных мощностей для решения сложных задач.

Однако работа в сети выдвигает и целый ряд проблем. К ним относятся, например:

- сохранность ценной информации общего использования;
- обеспечение надежности работы сетевой аппаратуры и сетевых программ;
- ограничение доступа к конфиденциальной информации;
- защита от компьютерных вирусов — вредоносных программ, наносящих различный ущерб аппаратуре и другим программам;
- разрешение конфликтов, когда несколько пользователей одновременно пытаются использовать одну и ту же аппаратуру, одни и те же программы или данные и т. д.

Несмотря на то что компьютерные сети являются частным случаем вычислительных систем, их архитектура имеет свои специфические особенности, связанные в первую очередь с необходимостью организовывать последовательную, побитовую передачу данных на большие расстояния для множества пар компьютеров. Поэтому краткому обсуждению архитектуры компьютерных сетей посвящена отдельная часть учебника.

Глава 19
Линии связи

Совокупность устройств, обеспечивающих передачу сообщений в сети, называется **линией связи**, или **приемно-передающей системой**. Любая линия связи включает в себя источник и приемник данных, кодирующие/декодирующие устройства, различные преобразователи и **канал** связи.

ВНИМАНИЕ

Совокупность устройств, которые используются для передачи сообщений между источником и приемником информации, называется линией связи. Протяженная в пространстве среда, через которую осуществляется передача сообщения, называется каналом связи, или просто каналом. Канал является важнейшей составной частью линии связи.

Не следует путать понятия *линия связи* и *канал связи*. Линия связи — более общее понятие, так как она включает в себя, кроме канала, ряд устройств, обеспечивающих передачу информации. Вместе с тем в устной и письменной речи эти термины используются как синонимы. Данная ситуация вполне аналогична почти равноправному использованию терминов «информация», «сообщение» и «данные». В учебнике термины «линия связи» и «канал связи» различаются только в тех случаях, когда это имеет принципиальное значение.

Различают *внутренние* (проходящие внутри компьютера) и *внешние* (проходящие вне компьютера) линии связи. Внешние линии связи отличаются от внутренних значительно большей пространственной протяженностью. Известно, что внутренние линии компьютера реализуются в виде различных шин. В принципе, к внутренним линиям можно отнести и кабельные соединения периферийных устройств компьютера с системным блоком, так как они имеют очень маленькую пространственную протяженность.

В качестве внешних линий связи используются:

- отдельно прокладываемые кабельные линии, которые состоят из одного или нескольких проводников, заключенных в один или несколько слоев изоляции, обеспечивающей электромагнитную, механическую, климатическую и другие виды защиты;

- существующие стандартные телефонные линии связи;
- волоконно-оптические линии связи;
- беспроводные радиоканалы наземной и спутниковой связи.

Различные виды линий обеспечивают различную дальность связи и имеют разную стоимость. Универсального, подходящего во всех случаях вида линий связи на сегодняшний день не существует.

19.1. Передача сообщений по линиям связи

Передача сообщений по линиям связи может осуществляться в различных режимах и различными способами. В этом разделе кратко обсуждаются особенности некоторых основных способов передачи.

19.1.1. Режимы передачи сообщений

Существует три режима передачи сообщений по последовательным каналам. Простейшим является **симплексный** режим, при котором передача возможна только в одном направлении — от источника к приемнику. Это очень простой и дешевый вариант связи, но полноценный обмен сообщениями в симплексном режиме невозможен. **Полудуплексный** режим обеспечивает обе стороны возможностью передавать сообщения, но только по очереди. Сначала линия связи предоставляется для передачи сообщения в одном направлении, а после ее завершения по той же линии организуется передача в другом направлении. Наиболее широкие возможности предоставляет **дуплексный** режим, при котором по одной и той же линии связи может осуществляться одновременная передача сообщений в двух направлениях.

19.1.2. Параллельная и последовательная передачи

Как было установлено ранее, существует два способа передачи информации в компьютерных линиях связи: *последовательная* и *параллельная*. Во время параллельной передачи каждый бит кода передается по отдельному проводнику, причем все биты передаются одновременно.

Недостатки параллельного способа передачи информации:

- невозможность передачи информации на большие расстояния, так как в результате взаимного влияния возникают искажения в отдельных проводниках шины, значительно увеличивающиеся при увеличении расстояния;
- неприемлемо высокая стоимость линии связи, состоящей из большого количества проводников.

Параллельный способ передачи используется в основном во внутренних линиях компьютера, а также для связи с некоторыми внешними устройствами, например с принтером.

Во время последовательной передачи все биты передаются по одному и тому же проводнику последовательно друг за другом. При этом линии связи имеют приемлемую стоимость, отсутствуют существенные ограничения на дальность передачи. Подавляющее большинство внешних линий связи реализуют последовательную (побитовую) передачу кода.

Переход от внутреннего параллельного кода к внешнему последовательному коду осуществляется с помощью *асинхронного преобразователя*, который обеспечивает преобразование *n* битов, одновременно передаваемых по разным линиям шин компьютера, в последовательность из *n* битов, передаваемых друг за другом по внешней линии связи (см. рис. 3.13).

Любые способы передачи сообщений должны обеспечить одновременность передачи и приема кода. Принимающий компьютер или устройство должны «знать», когда следует начинать считывание передаваемых данных и когда передача закончена. Отсутствие одновременности означает, что какая-то часть передаваемых данных будет пропущена или же в сообщение попадет какая-то часть «чужого» кода. Это может привести к частичному или полному искажению или утере переданного сообщения.

При передаче данных по внутренним линиям компьютера проблема одновременности решается синхронизацией приемного и передающего устройств компьютера с помощью синхроимпульсов одного и того же тактового генератора. При передаче сообщений от одного компьютера к другому по внешним линиям такая синхронизация затруднена, так как каждый из компьютеров имеет собственный тактовый генератор.

Для решения проблемы одновременности при передаче по внешним линиям разработано несколько способов. В частности, различают **синхронную** и **асинхронную** последовательные передачи. При синхронной передаче источник и приемник сообщения должны быть синхронизированы (совмещены) во времени по отправлению и приему каждого бита. Синхронизация осуществляется от одного и того же устройства. Она может производиться по отдельной линии, передающей и источнику, и приемнику тактовые импульсы от одного и того же генератора, которые синхронизируют передачу и прием кода. Синхронный прием и передача напоминают игру нескольких музыкальных инструментов в оркестре под управлением дирижера.

При асинхронной передаче источник и приемник не синхронизируются, поэтому дополнительная линия не нужна. Передача каждой очередной порции информации может начинаться в любой момент времени. Чтобы оповестить приемник о начале и об окончании передачи в передаваемый код включаются специальные служебные биты или байты, служащие ориентиром для приемника. Такие системы кодирования сообщений называются **самосинхронизирующимися**. Поскольку асинхронным способам передачи не требуется дополнительная линия связи, они дешевле, чем синхронные.

Существуют и другие режимы передачи, но они используются реже. В частности, можно упомянуть **изохронную** передачу, при которой передача и прием происходят в одни и те же строго определенные моменты времени, а также **плезиохронную** (то есть почти синхронную), когда источник и приемник обеспечиваются отдельными собственными источниками синхронизации, имеющими практически совпадающие частоты.

19.1.3. Способы представления кодов

Для представления передаваемых по линиям связи битов данных применяются **потенциальное** и **импульсное** кодирование, а также **модуляция** сигнала.

При использовании потенциального кодирования цифре 0 двоичного кода соответствует низкий уровень напряжения $U_н$, например 1 В, а цифре 1 — высокий $U_в$, например 5 В (рис. 19.1, *а*). В современных способах кодирования уровни напряжения гораздо ниже. Например, низким считается напряжение порядка 0,4–0,5 В, а высоким — более 2,4 В. Потенциальное кодирование обеспечивает хорошую помехоустойчивость, но, к сожалению, не обладает свойством самосинхронизации, так как при передаче, например, серии подряд идущих нулей напряжение в линии связи не изменяется. Поэтому приемник только по входному сигналу не может определить момент времени, соответствующий началу передачи. Кроме того, при высоких скоростях обмена и передаче серии подряд следующих нулей или единиц малейшая рассогласованность в тактовых частотах приемной и передающей сторон может привести к ошибке в один или более тактов.

В методе **биполярного** кодирования используются три уровня напряжения: низкий для цифры 0, а положительное или отрицательное высокое напряжение для цифры 1. В случае передачи нескольких подряд единиц каждая соседняя единица передается с помощью напряжения разных знаков. В этом подходе проблема передачи последовательности единиц решена, однако проблема передачи последовательности нулей остается.

В методе **манчестерского** кодирования время передачи одного бита делится на два равных по длительности участка. Цифра 0 кодируется сменой в середине такта низкого напряжения на высокое (рис. 19.1, *б*). Цифра 1 кодируется сменой в середине такта высокого напряжения низким. Этот вариант кодирования обладает хорошей самосинхронизацией для любых кодов, но требует удвоенной частоты передачи.

В схеме манчестерского кодирования предусмотрена возможность использования передачи двух специальных сигналов, в которых не производится смена напряжения в середине такта. Такие сигналы принято называть **запрещенными** кодами *J* и *K*. Запрещенный код *J* представляет собой удерживание низкого напряжения в течение всего такта, то есть этот сигнал фактически совпадает с кодированием нуля в потенциальной схеме. Запрещенный код *K* представляет собой удерживание высокого напряжения в течение всего такта, то есть он фактически совпадает с кодированием единицы в потенциальной схеме. Запрещенные сигналы включают в символы ограничители полезной информации.

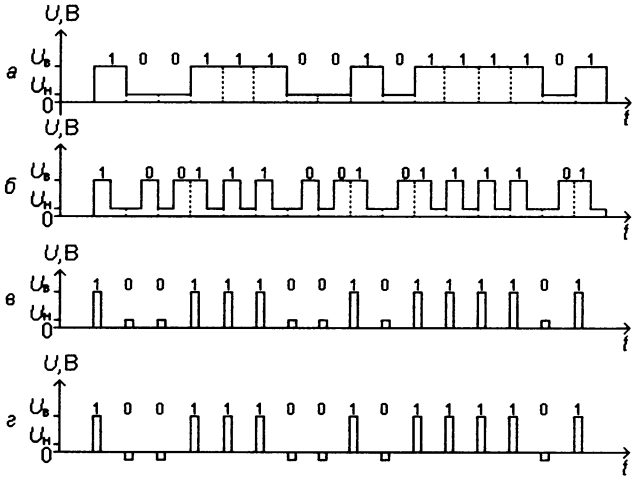


Рис. 19.1. Некоторые способы представления битов в линиях связи

При использовании импульсного кодирования нулю соответствует кратковременный импульс низкого напряжения, а единице — также кратковременный импульс высокого напряжения. При этом импульсы могут быть одной и той же полярности (рис. 19.1, *в*) или разных полярностей (рис. 19.1, *г*). Методы импульсного кодирования отличаются хорошей самосинхронизацией. Отметим, что потенциальное и импульсное кодирование может применяться в высокоскоростных линиях связи.

В низкоскоростных, телефонных линиях для передачи кода используется **модуляция** сигнала, которая представляет собой преобразование дискретного цифрового сигнала, используемого внутри компьютера, в аналоговый сигнал, используемый в телефонных линиях.

Сущность модуляции состоит в следующем. Зависимость напряжения электрического поля E от времени t в телефонной линии или в радиоканале описывается соотношением $E = A\sin(\varpi t + \varphi)$, где A — амплитуда, ω — частота и φ — фаза поля. Если параметры A, ω и φ в этом соотношении являются постоянными величинами, то эта зависимость описывает **несущий сигнал** (рис. 19.2, *а*), с помощью которого невозможно передать какую-либо информацию. Такой сигнал постоянно присутствует в телефонных линиях в отсутствие сообщений.

Модуляция представляет собой изменение во времени значения одного из параметров несущего сигнала, которое осуществляется в зависимости от заданного двоичного кода. Такое изменение, принятое на другом конце телефонной линии, обеспечивает однозначное восстановление исходного кода. В соответствии с выбранным для изменения параметром сигнала различают **амплитудную**, **частотную** и **фазовую** модуляцию.

На рис. 19.2, *б* показан график зависимости от времени напряжения для последовательного цифрового кода, поступающего из компьютера на вход **модема** (от модулятор/демодулятор) — устройства, обеспечивающего ту или иную разновидность модуляции. При использовании амплитудной модуляции для кодирования

цифры 0 амплитуда несущего сигнала устанавливается равной нулю или некоторой маленькой величине, а для кодирования цифры 1 для амплитуды выбирается большое значение. На рис. 19.2, *в* показан график зависимости от времени модулированного по амплитуде сигнала. Аналогично для передачи двоичного кода можно использовать скачкообразные изменения частоты ω и фазы φ несущего сигнала, получая частотную и фазовую модуляцию соответственно. Например, для осуществления фазовой модуляции используется изменение фазы сигнала на 180° при переходе от кода нуля к коду единицы или наоборот.

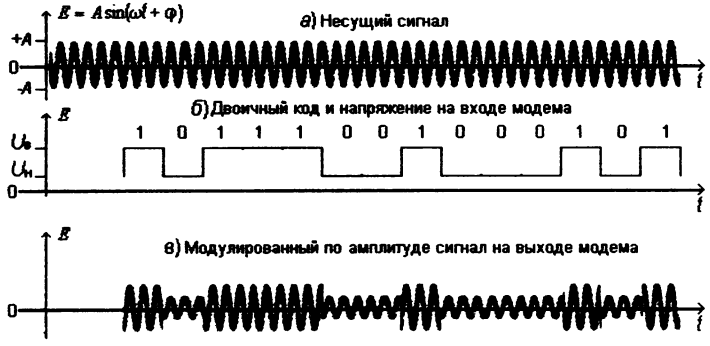


Рис. 19.2. Амплитудная модуляция сигнала

Существуют системы модуляции, в которых для кодирования используется более двух уровней амплитуды, частоты или фазы. Например, в начале временно́го отрезка, отводимого на передачу одного импульса, можно сдвигать фазу на 45, 135, 225 или 315°. Каждый такой сдвиг считается кодом сразу двух последовательных битов 00, 01, 10 или 11, передаваемых за время одного импульса. Этот способ кодирования называется **дибитовой** фазовой кодировкой, так как за один импульс передаются сразу два соседних бита кода.

Современные модемы используют комбинированные системы модуляции. Чаще всего комбинируются амплитудная и фазовая модуляции. Например, метод квадратурной амплитудной модуляции **QAM-64** (от Quadrature Amplitude Modulation) использует 8 уровней амплитудной модуляции и 8 уровней фазовой. При этом имеется 64 различных комбинаций уровней, которые позволяют закодировать в одном передаваемом импульсе 6 ($2^6 = 64$) соседних битов кода. Существуют и более мощные системы кодирования, однако нужно иметь в виду, что чем больше уровней вводится, тем сильнее влияние помех, так как расстояния между уровнями становятся все меньше и меньше и, следовательно, даже слабые помехи приводят к искажению кодов. В этом случае системы кодирования предусматривают включение в исходный код контрольных разрядов, обеспечивающих обнаружение и исправление ошибки.

Для обеспечения контроля правильности передачи, а также для обеспечения возможности работы в сети нескольких компьютеров одновременно последовательность передаваемых по сети битов принято делить на группы, которые называются **кадрами**.

ВНИМАНИЕ

Кадром называется совокупность битов, передаваемых по последовательной линии связи за один сеанс. Кадр — это самостоятельная единица передачи данных по последовательной линии связи.

Кадр наряду с передаваемыми информационными битами, то есть «полезными» данными, содержит различную служебную информацию: адрес получателя, адрес отправителя, контрольные биты, обеспечивающие обнаружение или исправление ошибок, и т. д. Общая длина кадра обычно не превышает 4 Кбайт.

Существует много различных способов формирования кадров. На рис. 19.3 в качестве примера изображена схема кадра одного из вариантов последовательной асинхронной передачи (так называемый стартстопный метод), который включает один стартовый бит в начале и один стоповый бит в конце кадра.

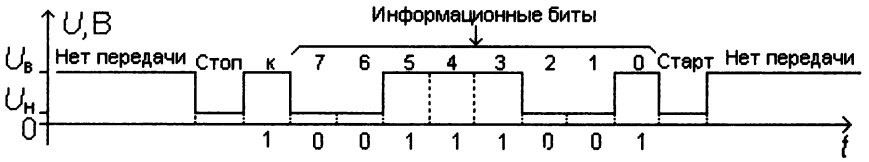


Рис. 19.3. Схема кадра последовательной асинхронной передачи

В отсутствии передачи в линии поддерживается высокое постоянное напряжение $U_в$. Передача кадра может начаться в любой момент времени. Она начинается с генерации стартового бита, который переводит линию в состояние нуля (точнее, низкого напряжения). Поскольку каждый бит кадра, как информационный, так и служебный, передается в течение одного и того же времени, по истечении этого интервала приемник «понимает», что сейчас начнется передача первого бита кода. После завершения передачи информационных битов передается контрольный бит четности. Завершает передачу стоповый бит (их может быть два), который переводит линию в состояние ожидания. Так как передача следующего кадра может начаться в любой момент времени после завершения передачи предыдущего кадра, наличие стопового бита дает компьютеру получателя возможность подготовиться к его приему. Включение в кадр стартовых и стоповых битов приводит к увеличению избыточности кода и общего времени передачи сообщения, но обеспечивает надежность передачи.

19.1.4. Обнаружение и исправление ошибок

Внешние линии, имеющие принципиально большую протяженность в пространстве, проходят вне экранированного корпуса компьютера по среде, создающей большое количество сильных электромагнитных помех, которые искажают передаваемые по линиям сигналы. Для повышения надежности и правильности передачи по линиям связи используются методы, основанные на включении

в передаваемый код некоторого количества дополнительных, контрольных битов, которые обеспечивают либо обнаружение, либо исправление ошибок. При этом большая часть функций контроля возлагается на сетевое программное обеспечение, так как на физическом, аппаратном уровне распознавание ошибок передачи затруднено.

В сильно зашумленных беспроводных и низкоскоростных линиях связи или в случаях, когда задержки на повторную передачу недопустимы, например при передаче мультимедийных данных, требуются системы кодирования, которые обеспечивают исправление ошибок. Наиболее широко распространены системы с исправлением ошибок, основанные на использовании кодов Хемминга (см. 2.6).

В высокоскоростных каналах связи, особенно в случае относительно невысоких требований к времени задержки ответа, используются коды, обеспечивающие только обнаружение ошибок, которые дают меньшую избыточность передаваемого кода. При таком подходе после обнаружения ошибки принимающая сторона запрашивает у передающей повторную передачу ошибочного кадра.

Для обнаружения ошибок наряду с включением контрольного бита в каждый передаваемый байт широко используется метод **контрольных сумм**, известный также как **полиномиальный**, или **CRC**-код (от Cyclic Redundancy Check — циклический избыточный контрольный). Суть этого метода состоит в том, что в передаваемый код включается вычисленная по специальному алгоритму целочисленная функция исходного кода, которую принято называть контрольной суммой. Код вместе с контрольной суммой образуют передаваемый кадр. Принимающей стороне известно положение в кадре и количество битов контрольной суммы, что создает возможность ее отделения от исходного кода. После разделения контрольной суммы и исходного кода принимающая сторона по тому же самому алгоритму повторно рассчитывает контрольную сумму. Если принятая и рассчитанная контрольные суммы не совпадают, значит, кадр содержит ошибки и следует запросить его повторную передачу.

19.2. Характеристики линий связи

Технические характеристики каналов связи определяют возможные области их использования. Основными характеристиками каналов являются:

- ширина полосы пропускания;
- минимальная длительность импульса;
- пропускная способность;
- скорость передачи информации;
- максимальная дальность.

Любые протяженные в пространстве физические системы, в том числе каналы связи, обладают определенными возможностями по передаче входного воздействия, осуществляемого в виде колебаний или импульсов. Если частота входного

воздействия оказывается слишком большой, то из-за наличия инерции система может не успевать за воздействием. Слишком медленные воздействия приводят к тому, что начальные участки системы быстро переходят в равновесное состояние, демпфируют воздействие, и оно просто не попадает на другую сторону системы. Это означает, что слишком большие или слишком маленькие частоты не проходят через систему. Приведенные общие рассуждения показывают, что для любой физической системы, в том числе для любых каналов связи, существует некоторый интервал частот, которые могут без искажения передаваться через систему.

ВНИМАНИЕ

Шириной полосы пропускания канала связи называется интервал частот ν, $\nu_{min} \le \nu \le \nu_{max}$, которые могут без искажения передаваться по каналу.

Например, органы слуха человека воспринимают колебания с частотой от $\nu_{min} = 16$ Гц до $\nu_{max} = 20\,000$ Гц, в то время как полоса пропускания телефонной связи значительно уже: от $\nu_{min} = 300$ Гц до $\nu_{max} = 3400$ Гц. Поэтому во время телефонных разговоров заметны искажения голосов собеседников.

Для телефонной связи ширина полосы пропускания определяется материалом мембраны, воспринимающей голос человека, свойствами порошка, сжимаемого мембраной, и т. д. Для электрического кабеля, используемого в компьютерных линиях связи, ширина полосы пропускания определяется его электрическими характеристиками: сопротивлением, пространственной протяженностью, распределенной емкостью, которая в электрических системах играет роль, аналогичную роли массы в механических, и т. д. Ширина полосы пропускания компьютерных линий связи значительно превосходит ширину полосы пропускания телефонных линий. В электрических кабельных линиях она достигает нескольких гигагерц. А ширина полосы пропускания оптоволоконных линий достигает десятков тысяч гигагерц.

Для любого канала связи существует минимальное время, за которое может быть передано сообщение. Быстрее, чем за это время, передать сообщение принципиально невозможно. Это время, так же как и ширина полосы пропускания, определяется физическими характеристиками канала. Например, для почты это время определяется расстоянием между точками отправления и приема почтового отправления, а также используемым транспортом — автомобиль, поезд, самолет и т. д.

Из-за высокой емкости протяженных линий связи нарастание переднего фронта импульса происходит довольно медленно (рис. 19.4, *а* и *б*). Если передавать импульсы с большей скоростью, то передний и задний фронты соседних импульсов могут искажать друг друга, и тогда возникнут сложности в отделении друг от друга двух соседних импульсов (рис. 19.4, *в*). Именно этими факторами с физической точки зрения обусловлено наличие *минимального времени передачи импульса* τ_0 в компьютерных линиях связи.

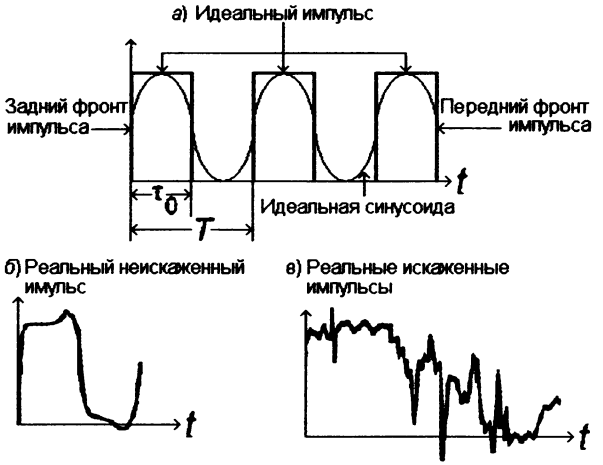


Рис. 19.4. Импульсы в линиях связи

При передаче двоичных кодов по каналам связи в качестве параметра, характеризующего время передачи сообщения, выступает период импульса. Напомним, что период — это отрезок времени, в течение которого происходит одно полное колебание, а частота — это количество колебаний в единицу времени. Следовательно, минимальный период T определяется как величина, обратная максимальной частоте полосы пропускания ν_{max}. Длительность элементарного импульса τ_0, с помощью которого передается одна двоичная цифра сообщения, равна половине периода сигнала (рис. 19.4, *а*):

$$\tau_0 = \frac{T}{2} = \frac{1}{2\nu_{max}}.$$

Пропускная способность канала определяется как отношение передаваемого с одним импульсом количества информации к времени его передачи τ_0. **Скорость передачи** информации по каналу определяется как отношение количества переданной информации ко времени, в течение которого оно передано. Пропускная способность является предельно возможной характеристикой канала и вычисляется на основании времени передачи одного импульса τ_0. Скорость передачи информации представляет собой усредненную характеристику передачи данных в той или иной конкретной ситуации и вычисляется по произвольному отрезку времени. Максимальная скорость передачи информации по каналу связи равна его пропускной способности.

Скорость передачи данных по линиям связи и пропускную способность канала выражают либо в количестве импульсов, либо в количестве битов, передаваемых по линии в единицу времени. В первом случае единицей скорости является **бод**, который равен одному импульсу, передаваемому по линии связи за одну секунду. Если с помощью одного импульса передается код одной двоичной цифры, как это происходит в простейших системах модуляции, то 1 бод = 1 бит/с. Если используются специальные системы модуляции типа дибитовой фазовой кодировки

или системы кодирования QAM-64, то измеряемая в битах в секунду скорость передачи оказывается в соответствующее число раз больше скорости, измеренной в бодах. Например, известно, что максимальная скорость передачи по телефонным линиям равна 2400 бод. Если с одним импульсом передается, скажем, 6 бит (как в системе QAM-64), то скорость передачи 2400 бод = 14 400 бит/с. Существуют системы модуляции и модемы, обеспечивающие при одной и той же пропускной способности линии связи 2400 бод передачу со скоростью 33 600 и 56 000 бит/с.

Верхняя граница полосы пропускания стандартных телефонных линий связи ν_{max} = 3400 Гц ограничивает их пропускную способность значением 2400 бод, а связанную с ней максимальную скорость передачи — значением 56 000 бит/с. Эта верхняя граница выбрана из соображений оптимизации передачи человеческой речи. Для получения указанной верхней границы полосы пропускания в передающую аппаратуру телефонных линий связи включаются специальные фильтры.

Чтобы лучше приспособить телефонные линии к потребностям компьютерных сетей и увеличить скорость передачи, был предложен стандарт **ADSL** (от Asymmetric Digital Subscriber Line — асимметричная цифровая абонентская линия). Этот стандарт основан на исключении таких фильтров для отдельных телефонных линий, используемых в компьютерных сетях. Стандарт ADSL обеспечивает пропускную способность 4000 бод. Это дает возможность приспособленным для работы по этому стандарту ADSL-модемам принимать сообщения со скоростью до 8 Мбит/с и отправлять их со скоростью до 1 Мбит/с.

Пропускная способность электрических кабельных линий связи достигает нескольких мегабит в секунду, а у оптоволоконных линий связи современный предел пропускной способности равен 10 Гбит/с.

Любые передаваемые по линиям связи сигналы подвержены затуханию, возрастающему по мере увеличения проходимого сигналом расстояния. Это затухание вызвано рядом физических факторов, в том числе сопротивлением передающей среды. Затухание выражается в уменьшении амплитуды сигнала. Если уровень амплитуды сигнала становится сравнимым с уровнем шума, то прочитать пересылаемое сообщение невозможно. Поэтому при передаче на дальние расстояния осуществляется периодическое восстановление исходных амплитуды и мощности сигнала с помощью различных дополнительных устройств. **Максимальной дальностью** передачи по линии связи считается расстояние, на которое по ней может быть передан код без использования вспомогательных средств. Для разных сетей эта характеристика находится в диапазоне от сотен метров до десятков тысяч километров.

Для характеристики нагрузки на передающие линии сети используется понятие **трафика**. Трафиком называется количество сообщений, передаваемых по линии связи в единицу времени. Другими словами, трафик представляет собой поток сообщений, рабочую нагрузку линий связи. Трафик в компьютерных сетях является аналогом загруженности железнодорожной ветки, которая выражается в количестве составов, проходящих по ней за единицу времени.

Контрольные вопросы и упражнения

1. Дайте определение компьютерной сети и укажите основные характерные особенности, отличающие сеть от других классов вычислительных систем.
2. Сформулируйте основные преимущества, которые получает пользователь, работающий в компьютерной сети.
3. Сформулируйте основные проблемы, возникающие при организации компьютерных сетей.
4. Чем отличается линия связи от канала связи?
5. Перечислите основные разновидности каналов связи.
6. Охарактеризуйте существующие режимы передачи по линиям связи.
7. Перечислите присущие параллельному способу передачи данных недостатки, препятствующие его применению в качестве внешних линий связи.
8. Охарактеризуйте используемые на практике способы представления цифровых кодов в линиях связи.
9. Опишите способ манчестерского кодирования. В чем его достоинства и недостатки?
10. Как осуществляется передача цифровых кодов по телефонным линиям связи?
11. Какие способы модуляции вам известны?
12. Что представляют собой комбинированные способы модуляции? В чем их достоинства и недостатки?
13. Что представляет собой кадр? Изобразите схему стартстопного кадра.
14. Какие способы используются для обеспечения надежности передачи сообщений по компьютерным сетям?
15. Дайте определения основных технических характеристик линий связи.
16. Что такое полоса пропускания и чем вызвано ее появление?
17. Чем отличается пропускная способность от скорости передачи?

Глава 20
Классификация и топология сетей

Компьютерные сети классифицируются по дальности охвата территории и, следовательно, по способу использования и количеству подключаемых компьютеров, а также по функциональной структуре. Кроме того, выделяют проводные и беспроводные сети. Указанные классификационные схемы являются общезначимыми. Вместе с тем существует большое количество градаций сетей, так или иначе связанных с вопросами технического характера, со способом их функционирования. Наиболее важные из них обсуждаются в последующих главах учебника.

По дальности охвата территории выделяют:

- домашние сети;
- локальные сети, или LAN (от Local Area Network);
- городские сети, или MAN (от Metropolitan Area Network);
- территориальные (региональные) сети, или WAN (от Wide Area Network);
- глобальные сети.

Категория домашних сетей появилась относительно недавно. Домашняя сеть может объединять несколько компьютеров, используемых в одной квартире, доме, семье. Однако это не главный момент в выделении обсуждаемой категории. Более важным является подключение к сети, центральным элементом которой является компьютер, множества других используемых в быту цифровых (и не только) устройств. В частности, к домашней сети могут подключаться различные звуко- и видеозаписывающие и воспроизводящие устройства — музыкальные центры, телевизоры, видеокамеры. Другой важной группой устройств, которые могут быть подключены к домашней сети, являются средства связи — телефоны, факсы, видеотелефоны, а также устройства, обеспечивающие связь по Интернету. Всевозможные бытовые устройства: холодильники, микроволновые печи, отопительные системы, кондиционеры, счетчики расхода электроэнергии и т. д. —

после подключения к домашней сети могут управляться компьютером с помощью специализированных программ. Более того, информацию из домашней сети можно передавать по локальной или глобальной сети в расчетные центры и автоматизировать системы расчета за бытовые услуги. Предполагается, что домашние компьютерные сети станут со временем такими же привычными, как и другие домашние сети — водопроводная, отопительная и т. д.

Локальные сети обеспечивают работу специалистов лаборатории, отдела, небольшого предприятия, расположенного в одном здании или в группе близко находящихся зданий. Локальная сеть может включать несколько сотен компьютеров. В локальных сетях, как правило, используются однотипные аппаратура и программное обеспечение.

Попутно заметим, что сеть, построенную из одинаковых, однотипных или совместимых друг с другом компьютеров, называют **однородной (гомогенной)**. Если же сеть содержит разнотипные, несовместимые аппаратно либо программно компьютеры, то она называется **неоднородной (гетерогенной)**.

Городские сети объединяют локальные сети различных предприятий и организаций, находящихся, как правило, в пределах города или сопоставимого по площади района. К уровню городских сетей относятся модемные системы удаленного доступа домашних компьютеров к провайдерам, которые предоставляют им выход в Интернет.

Территориальные или региональные сети объединяют локальные или городские сети, расположенные в различных городах или районах. Такие сети могут охватывать несколько областей, а также целые страны и континенты. Примеры территориальных сетей России: Relcom (от Reliable Communications — надежные коммуникации), Relarn (от Russian Electronic Academic Research Network — российская академическая исследовательская электронная сеть), Rosnet, Sprint, Unicom, Роспак и ряд других. Примеры территориальных сетей других стран — JANET (Великобритания), DFN (Германия), Nordunet (Скандинавские страны), EUNet (от European Unix Users Network — Европейская пользовательская Unix-сеть), EARN (European Academic and Research Network — Европейская академическая исследовательская сеть), CompuServe и America Online (американские коммерческие информационные сети).

По мере роста и развития территориальных сетей возникали устойчивые соединения и связи между ними. Так появились глобальные сети, сети планетарного масштаба, охватывающие все континенты Земли, например сети Usenet, Fidonet, Интернет. Территориально разделенные фрагменты глобальных сетей соединяются друг с другом с помощью спутниковой связи, радиосвязи, телефонных линий и оптоволоконных каналов.

Функциональная классификация сетей основана на особенностях использования и работы различных компьютеров, подключенных к сети, на их отношении к различным программным и аппаратным ресурсам. Ранее отмечалось, что все устройства компьютера (процессор, дисплей, принтеры, дисковые накопители и т. д.), а также программы и данные, находящиеся на накопителях, образуют его ресурсы.

Ресурсы компьютера, к которым разрешено обращение от других компьютеров сети, называются сетевыми ресурсами. Такими устройствами чаще всего являются диски (сетевые диски) и принтеры (сетевые принтеры). Ресурсы компьютера, недоступные для других компьютеров сети, называются локальными.

Сеть, в которой все компьютеры имеют совершенно равные права и могут использовать сетевые ресурсы друг друга, называется **одноранговой**. Обычно в одноранговую сеть объединяется небольшое количество компьютеров (в пределах нескольких десятков), на которых работает группа сотрудников, занятых решением одной и той же задачи. Каждый из сотрудников, работая на любом компьютере сети, может использовать сетевые ресурсы любого другого компьютера. Например, работая с одним компьютером, можно просматривать и изменять документы, находящиеся на дисках второго компьютера, и пересылать их на принтер, подключенный к третьему. Пользователь практически не ощущает разницы между работой с устройствами (дисками, принтерами), принадлежащими его собственному компьютеру, и работой по сети с устройствами соседа.

В более развитых сетях существует специализация компьютеров. Одни компьютеры, как правило, более мощные, предоставляют свои ресурсы другим компьютерам, а последние используют эти ресурсы для своих целей. Такие сети относятся к типу **клиент-сервер**.

ВНИМАНИЕ

Сервером называется компьютер, который предоставляет свои сетевые ресурсы другим компьютерам.

Иногда в названии сервера указывается характер предоставляемых ресурсов или услуг. Например, почтовый сервер — сервер, предоставляющий почтовые услуги, то есть пересылающий сообщения между остальными компьютерами сети. К серверам предъявляются повышенные требования в смысле мощности и надежности их аппаратного и программного обеспечения, которое должно круглосуточно поддерживать работу сети. Сервер должен иметь высокую производительность, большой объем оперативной и дисковой памяти. В настоящее время в качестве серверов используются мощные персональные компьютеры, универсальные машины или суперкомпьютеры.

ВНИМАНИЕ

Клиентом называется компьютер или набор аппаратных средств, который использует в своей работе ресурсы другого компьютера.

Как правило, в качестве клиентов используются стандартные персональные компьютеры. В этом случае часть работы выполняется на клиентской машине, а при возникновении потребности в каком-либо дополнительном ресурсе происходит обращение к серверу. В некоторых случаях клиентом может быть простой терминал, то есть набор аппаратуры, состоящий только из клавиатуры и дисплея. В этом случае вся обработка информации выполняется на серверах, а пользователь, работая на клавиатуре, передает запросы к серверам и получает от них ответ на дисплее.

К категории **проводных** относятся сети, в которых в качестве каналов связи используются любые провода и кабели — телефонные, специальные электрические и оптоволоконные. В **беспроводных** сетях каналами связи являются электромагнитные колебания различных частот. Беспроводные сети могут иметь любой масштаб. Это могут быть домашние сети, в которых периферийные устройства обмениваются информацией с системным блоком с помощью радиосвязи. Аналогичным образом ноутбук может связываться со стационарным компьютером. Беспроводные локальные сети целесообразно развертывать в тех ситуациях, когда прокладка кабельных соединений нецелесообразна или невозможна, например, в полевых условиях. Беспроводные глобальные сети, основанные на спутниковой или иной разновидности радиосвязи, могут использоваться в географических зонах с неблагоприятными условиями, например в Антарктиде.

Топологией сети называется способ организации физических соединений между подключенными к ней компьютерами. В целом топология сетей является полным аналогом топологии многомашинных вычислительных систем. Терминалы, клиенты, серверы, коммутаторы и другие подключенные к сети устройства называют **узлами** сети, или **станциями**. Узлы сети, представляющие собой компьютеры, обеспечивающие выполнение программ пользователей (приложений), иногда называют **хостами** (от host — хозяин, ведущий).

Топология сети обычно представляется в виде графа, вершинами которого являются узлы сети, а в качестве ребер выступают физические линии связи между ними.

Топология глобальных сетей отличается неоднородностью и сложностью, в то время как топология локальных сетей отличается, как правило, однородным составом и четкой структуризацией. Топология сети определяет много важных ее характеристик. Например, наличие в графе резервных линий связи между узлами сети повышает ее надежность, дает возможность сбалансировать нагрузку.

В полносвязной топологии (см. рис. 17.2, *а*) каждый компьютер сети связан индивидуальной линией со всеми остальными узлами. К недостаткам этой топологии относятся плохая масштабируемость и высокая стоимость, так как нужен отдельный порт и отдельная линия для соединения каждого узла сети с каждым другим узлом. Однако полносвязная топология, по-видимому, имеет хорошие перспективы развития, так как она обеспечивает высокую эффективность передачи информации.

Топологии, в которых отсутствуют отдельные линии связи между какими-либо двумя узлами сети, называются неполносвязными. Обмен между не связанными узлами требует транзитной передачи данных через узлы, связанные линиями. Неполносвязные топологии гораздо дешевле полносвязных.

Важнейшей разновидностью неполносвязной топологии является **ячеистая** топология, которая образуется удалением из полносвязной топологии линий с низким объемом передаваемой информации. Такую топологию имеют, например, **подсети** глобальных сетей, упрощенная схема которых представлена на рис. 20.1. Подсеть состоит из большого количества линий связи и специализированных компьютеров, которые обычно называют **маршрутизаторами**. С маршрутизаторами подсети связаны локальные сети различных организаций и использующие модемную связь отдельные компьютеры пользователей. Основная задача подсети

состоит в доставке сообщения от компьютера-источника компьютеру-адресату. Поскольку подсеть имеет ячеистую топологию, не любые ее узлы связаны напрямую. Если сообщение передается между узлами, не имеющими прямой связи, то маршрутизаторы должны выбрать оптимальный путь его передачи. Подсети глобальных сетей занимают значительные территории, а подсеть Интернета покрывает всю поверхность Земли.

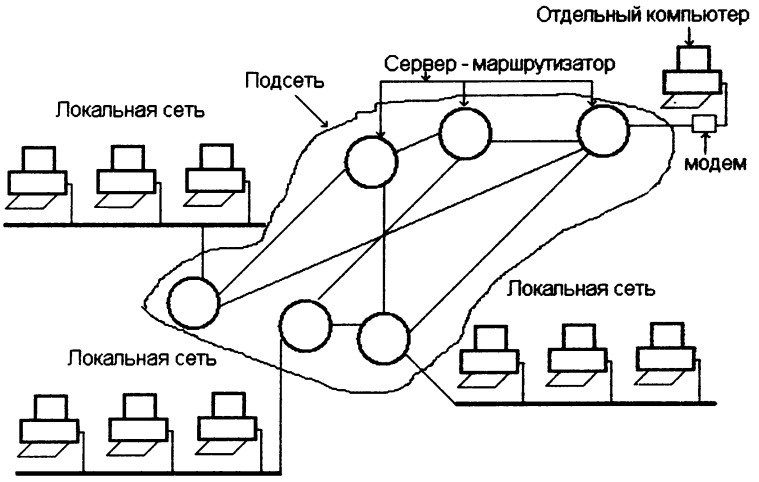


Рис. 20.1. Упрощенная схема глобальной сети

Одной из самых распространенных топологий локальных сетей небольшого масштаба считается топология с *общей шиной* (см. рис. 17.2, *б*). Ее основными достоинствами являются низкая стоимость монтажа, унифицированный способ подключения и мгновенный *широковещательный* режим передачи. К основным недостаткам шинной топологии относятся ее низкая надежность, связанная с тем, что любой дефект шины выводит из строя всю сеть целиком, а также невысокая пропускная способность — в любой момент времени только один компьютер может передавать сообщения по шине.

В другой распространенной топологии локальных сетей, типа *кольцо* (см. рис. 17.2, *в*), сообщения передаются по замкнутым в кольцо линям связи от одного узла к другому. Это существенно отличает кольцевую топологию от шинной, в которой используется широковещательный режим передачи сообщений. Основные недостатки кольцевой топологии те же, что и у шинной, — невысокие надежность и пропускная способность. Отметим, что основную линию связи в кольцевой и шинной топологиях обычно называют **сетевой магистралью**.

В топологии *звезда* (см. рис. 17.2, *г*) каждый компьютер подключается индивидуальной линией связи к общему мощному центральному узлу, который может передавать сообщения всем остальным узлам в широковещательном режиме или же одному конкретному адресату. Сообщения между концевыми компьютерами передаются только через центральный узел. Топологию «звезда» отличают более высокая надежность по сравнению с шинной топологией, а также возмож-

ность обеспечить высокую конфиденциальность с помощью контроля поступающих в центральное устройство сообщений. Основными недостатками «звезды» являются более высокая стоимость по сравнению с шинной топологией, низкая масштабируемость и очень высокие требования, которые предъявляются к центральному компьютеру «звезды». Чтобы увеличить эффективность этой топологи, функции центрального узла во многих случаях разделяют между сетевым сервером и коммутатором, которые совместно образуют центральный узел топологии.

Разновидностью топологии «звезда» является **древовидная** топология, или **иерархическая звезда** (рис. 20.2), которая считается самой популярной и распространенной в локальных и глобальных сетях. Это одна из самых отказоустойчивых топологий. Кроме того, она обычно имеет структуру, наиболее соответствующую реальным информационным потокам в сети.

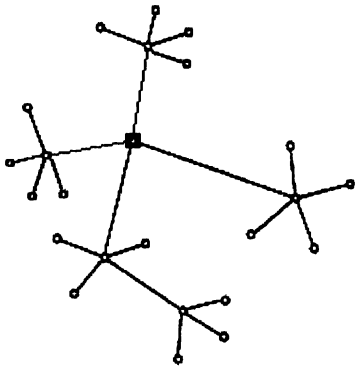


Рис. 20.2. Топология типа «иерархическая звезда»

На практике только небольшие локальные сети могут иметь типовую топологию. Более крупные локальные и, тем более, глобальные сети имеют смешанную топологию, их отдельные участки обладают типовой топологией. Другими словами, крупные реальные сети фактически являются объединением нескольких сетей меньшего масштаба с различными базовыми топологиями.

При объединении в сети более двух компьютеров возникает проблема указания адресата передаваемого сообщения. Адрес узла сети должен удовлетворять следующим требованиям:

- однозначная идентификация (распознавание) любого компьютера в сети любых масштаба и типа;
- сведение к минимуму вероятности дублирования имен;
- сведение к минимуму ручного труда операторов, администраторов и пользователей при задании адреса в сети;
- удобство работы с адресом для пользователей (например, для запоминания);
- желательна иерархическая структура адреса в сети (наподобие почтового адреса), в противном случае неизбежны временные издержки на передачу сообщения, а также дополнительные расходы памяти;

- желательна также компактная структура адреса, которая избавляет коммуникационную аппаратуру от излишних накладных расходов.

Анализ этих требований показывает, что в целом они взаимно противоречивы. Поэтому на практике используют несколько различных схем адресации, каждая из которых применяется в наиболее подходящих ситуациях. В необходимых случаях по специальным алгоритмам по адресу одного типа можно однозначно получить адрес другого типа.

Основными схемами адресации в сетях являются аппаратная, символьная и числовая. Аппаратная адресация основана на закреплении за каждым компьютером сети уникального (то есть нигде больше в данной сети не используемого) двоичного или шестнадцатеричного номера. Например, последовательность 009AF917718DCA700 может рассматриваться как шестнадцатеричный аппаратный адрес. Такая схема используется в малых и средних по количеству узлов сетях. Аппаратные адреса формируются и используются аппаратурой сети автоматически либо фиксируются на заводе-изготовителе. Недостатками аппаратной адресации являются отсутствие иерархической структуры и необходимость в серьезной перенастройке сети в случае замены некоторой части ее оборудования, а также при удалении или добавлении новых узлов.

Символьные адреса представляют собой обычные имена, например Newton, Star, которые администраторы локальных и глобальных сетей закрепляют за компьютерами. Эти имена удобны для использования людьми, так как обычно несут смысловую нагрузку. Кроме того, символьные имена могут иметь иерархическую структуру. Характерным примером является доменная адресация в сети Интернет. Так, адрес ssu.samara.ru относится к символьным составным, то есть иерархическим адресам.

Числовые составные адреса, например IP-адрес 219.09.154.21 в Интернете, также имеют иерархическую структуру, но строятся не из имен, а из чисел. Числовой адрес гораздо более компактный, чем символьный, поэтому его выгодно использовать в качестве передаваемого по сети вместе с сообщением. Но в то же время пользователю удобнее задавать символьный, доменный адрес. Чтобы согласовать эти противоречивые требования, люди используют доменную адресацию, а аппаратура сети — числовую. Преобразование доменного адреса в IP-адрес выполняется так называемым **DNS-сервером** (от Domain Name Server).

Контрольные вопросы и упражнения

1. Как классифицируются компьютерные сети по охвату территории?
2. Чем отличается однородная сеть от одноранговой?
3. Поясните смысл терминов «сервер» и «клиент».
4. Охарактеризуйте топологии, применяющиеся в компьютерных сетях.
5. Какие требования предъявляются к сетевому адресу? Охарактеризуйте способы адресации, используемые в компьютерных сетях.

Глава 21
Элементы сетевого оборудования

Аппаратные средства, которые обеспечивают физические аспекты обмена информацией в компьютерных сетях, включают в себя аппаратуру передачи данных — сетевые адаптеры, радиопередающие и радиоприемные средства, — а также различные элементы передающих сред: кабели, соединители, повторители, терминаторы и т. д.

Компьютер, подключаемый к локальной сети предприятия, должен быть укомплектован *сетевой* платой, которая вставляется в один из слотов системной платы, или автономно функционирующим *сетевым адаптером*. Собственно подсоединение к сети осуществляется с помощью электрического кабеля, один конец которого вставляется в находящийся на сетевой плате (адаптере) разъем, а второй подсоединяется к сетевой магистрали (рис. 21.1).

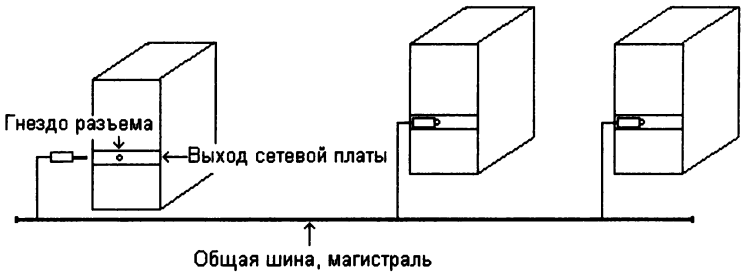


Рис. 21.1. Упрощенная схема подключения компьютеров к локальной сети

При использовании телефонных линий применяется другая схема подключения. Последовательный код поступает с выхода асинхронного преобразователя на модем, который должен быть подключен, например, к городской телефонной сети.

Модем преобразует поступивший из компьютера цифровой код в аналоговый сигнал, который обычным порядком передается по телефонной сети к другому компьютеру. На втором конце линии сигнал преобразуется в обратном порядке (рис. 21.2).

Рис. 21.2. Схема передачи кода по телефонной сети с использованием модема

В беспроводных сетях используются передатчики и приемники инфракрасного и лазерного излучения, а также радиопередача на одной или нескольких несущих радиочастотах. Различают несколько вариантов сетей, основанных на инфракрасном излучении. В частности, это сети с передачей в пределах прямой видимости источника и приемника, сети с передачей на рассеянном от стен и потолков помещений излучении, а также сети с отражением сигнала от специально установленных устройств. Инфракрасные сети обеспечивают передачу на расстояниях до 30 м со скоростью до 10 Мбит/с. Беспроводные сети, основанные на лазерном излучении, всегда используют передачу в пределах прямой видимости. При передаче по радиоканалу прямая видимость не является обязательным условием. Но скорость передачи значительно меньше, чем при использовании лазерных технологий. В радиоканалах диапазоны коротких, средних и длинных волн обеспечивают дальнюю связь при относительно невысокой скорости передачи данных. Более скоростными являются каналы ультракоротких и сверхкоротких волн.

Беспроводные сети обычно оснащаются устройствами, которые обеспечивают связь входящих в нее компьютеров с проводными сетями, в частности, для получения доступа в Интернет. Такие устройства, представляющие собой сетевой адаптер с передающим и принимающим радиоволны элементами, принято называть **точками доступа** беспроводной сети.

Основными элементами кабельных систем сети являются собственно кабели и различные средства их соединения. Используемые в сетях типы кабелей описаны в наборе стандартов с названием **10Base**. Кратко обсудим технические характеристики наиболее часто применяемых типов кабелей.

Электрический кабель, называемый **витой парой**, описывается стандартом **10Base-T**. Витая пара представляет собой два изолированных медных провода, скрученных для защиты от взаимовлияния и от влияния внешних помех. Несколько таких витых пар могут быть помещены в единую защитную оболочку. Различают

неэкранированные, или **UTP** (от unshielded twisted pair), и **экранированные**, или **STP** (от shielded twisted pair), витые пары. Кабель экранированной витой пары, в отличие от кабеля неэкранированной пары, имеет дополнительную медную оплетку, что обеспечивает повышенную защиту от помех. Достоинствами этого типа соединений являются низкая стоимость, простота прокладки, легкая перестраиваемость структуры сети. Витая пара в зависимости от ее типа обеспечивает возможность подсоединения в локальных сетях на расстояниях до 100 м и передачу данных со скоростью от 4 до 100 Мбит/с.

Стандарт **10Base-2** описывает **тонкий коаксиальный** кабель с диаметром центрального, как правило, медного, проводника 0,64 см (0,25 дюйма). К преимуществам этой разновидности кабелей относятся почти такие же удобство и простота прокладки, как у витой пары, при этом обеспечивается максимальная дальность соединения до 185 м и подключение до 30 узлов в сети. Тонкий коаксиальный кабель наряду с витой парой используют для подключения сетевого адаптера компьютера к сетевой магистрали (рис. 21.5).

Стандарт **10Base-5** описывает **толстый коаксиальный** кабель с диаметром центрального проводника около 1,27 см (0,5 дюйма). Он обеспечивает дальность передачи до 500 м и возможность подключения до 100 компьютеров. В то же время толстый кабель неудобен в прокладке, его сложно гнуть и подсоединять к другим элементам сети. На рис. 21.3 изображена схема подсоединения узлов сети к толстому коаксиальному кабелю. Центральный проводящий элемент кабеля заключен в два слоя изоляции и один экранирующий слой проводника, благодаря которому помехи не оказывают сильного влияния на передаваемый код. При этом большая толщина проводника обеспечивает низкое сопротивление и возможность передачи сообщений на дальние расстояния.

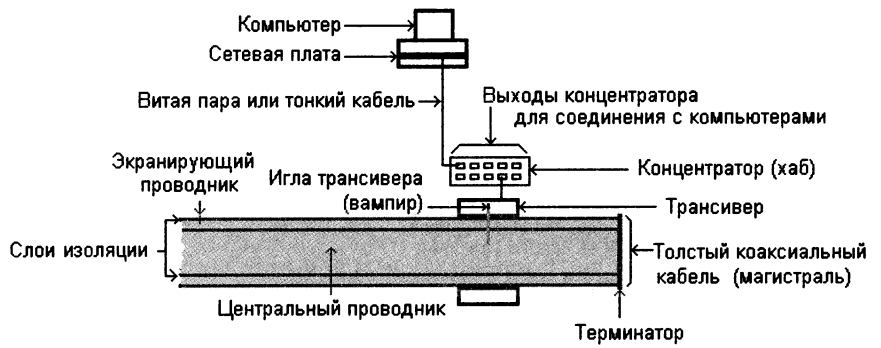


Рис. 21.3. Схема подключения к толстому коаксиальному кабелю

Для подключения к толстому кабелю используется так называемый **трансивер** (от transceiver — transmitter/receiver — передатчик/приемник), представляющие собой надеваемую на магистраль коробку, внутри которой имеется специальная игла, иногда называемая «**вампиром**». Она протыкает изолирующие слои кабеля и обеспечивает тем самым подключение к центральному проводнику. Такой

способ подключения к кабелю не нарушает его электрических свойств, которые очень важны для эффективного функционирования всей сети в целом. Далее к трансиверу подключается **концентратор**, или **хаб** (от hub — ступица), играющий роль разветвителя. Концентратор имеет от 4 до 16 выходов для подключения компьютеров и один вход, через который концентратор подключается к трансиверу. Компьютеры подключаются к концентратору с помощью витой пары или тонкого кабеля, один конец которого соединяется с сетевой платой компьютера, а второй крепится к концентратору.

Волоконно-оптический кабель, описываемый стандартом **10Base-F**, состоит из заключенных в несколько защитных слоев световодов, которые представляют собой прекрасно проводящие лучи света стеклянные волокна диаметром от 5 до 100 мкм. Проводящее волокно световода заключено в слой стекла с другими отражающими свойствами, которые не позволяют попавшим внутрь световода лучам света выйти за его пределы (рис. 21.4, *б*). Несколько гибких и легко повреждаемых световодов заключают внутрь прочной трубки, которая образует для них нечто вроде защитного каркаса. Такую конструкцию называют **волоконным узлом**. Например, волоконный узел, изображенный на рис. 21.4, *а*, состоит из четырех световодов. Несколько волоконных узлов вместе с прочным центральным элементом кабеля заключаются в один или несколько защитных слоев, которые обеспечивают механическую, термическую и другие виды его защиты.

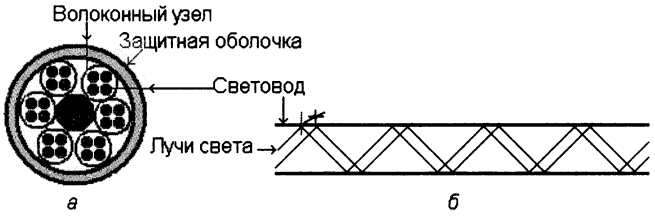


Рис. 21.4. Упрощенная схема оптоволоконного кабеля: *а* — поперечный разрез; *б* — прохождение лучей по волокну

Волоконно-оптические кабели обладают полосой пропускания от 500 МГц до 100 ГГц, обеспечивают дальность передачи до нескольких десятков километров при скорости до 10 Гбит/с. Это довольно дорогой вид линий связи, к тому же, требующий высокой квалификации при проведении монтажных работ, отличающихся более высокой сложностью по сравнению с прокладкой электрических кабелей. Оптоволоконные линии связи чаще всего используются в региональных и глобальных сетях.

Предполагается, что в перспективе существующие сейчас телефонные линии должны быть повсеместно заменены линиями **ISDN** (от Integrated Services Digital Network — цифровая сеть с интегрированными службами), основанными на цифровой форме представления данных. Линии ISDN обеспечивают дальность передачи до 5 км со скоростью до 2 Мбит/с при значительно более высоком качестве, чем у телефонных линий. При переходе к цифровым линиям связи

необходимость в модемах отпадает, так как форма сигнала в цифровой телефонной сети и в компьютере одинаковая — цифровая.

Цельный участок кабеля, обеспечивающий связь между компьютерами в локальной сети без использования вспомогательных устройств, принято называть **кабельным сегментом**. Для объединения различных сегментов кабельных систем используются различные **коннекторы** (от connector — соединитель), **Т-разветвители** и **терминаторы** (от terminator — ограничитель).

Различные виды коннекторов используются для сращивания, соединения двух сегментов кабеля, а также для подключения сегмента кабеля к Т-разветвителю, который имеет один разъем для его подсоединения к сетевому адаптеру и два гнезда для подключения сегментов кабелей от соседних узлов сети (рис. 21.5). Сегмент кабеля на конце должен иметь коннектор, который припаивается или механически обжимается к кабелю. Затем коннектор подсоединяется к гнезду Т-разветвителя.

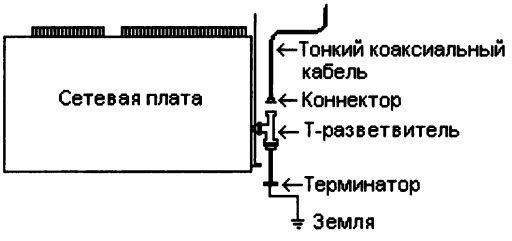


Рис. 21.5. Элементы кабельных соединений для подключения к сетевой плате

Терминатор представляет собой заземленную заглушку, устанавливаемую на свободном конце кабеля для отражения электрических импульсов внутрь проводника. При отсутствии терминатора поле на свободном конце кабеля рассеивается в окружающее пространство и связь между подключенными к кабелю компьютерами становится невозможной. Поэтому терминаторы должны быть поставлены на все свободные концы сетевых кабелей.

В организации функционирования сетей программное обеспечение играет не менее важную роль, чем аппаратное. В общем случае принадлежащие различным элементам сети модули программного обеспечения обеспечивают выполнение всех действий, необходимых для организации передачи данных. Программные модули, реализующие указанные функции, входят в состав сетевых операционных систем, таких как Novell Netware, Sun Solaris, Unix, Windows NT и т. д., или организованы в самостоятельные программные системы.

Важнейшей составной частью сетевого программного обеспечения являются **сетевые службы**, которые предоставляют пользователям возможность выполнять нужные им действия: отправлять и получать электронные письма, посылать запросы в базу данных, находящуюся на компьютере в другой стране, копировать файлы с удаленного компьютера, не заботясь при этом о множестве технических деталей этих процессов. Сетевые службы предоставляют пользователям все эти

и другие возможности, обеспечивая при этом удобный и простой стиль работы. Сетевые службы реализуются в виде отдельных программных систем или в виде утилит, подключаемых к операционным системам. Кроме обслуживания потребностей пользователей, на сетевые службы возлагается масса других задач. В частности, это задачи организации распределенных вычислений, администрирования, обеспечения надежности и непротиворечивости хранения данных в распределенных базах и хранилищах данных.

Контрольные вопросы и упражнения

1. Поясните смысл терминов «сетевой адаптер», «модем», «точка доступа».
2. Сравните между собой возможности витой пары, тонкого и толстого коаксиальных кабелей и оптоволоконного кабеля.
3. Поясните смысл терминов «коннектор», «Т-разветвитель», « терминатор».
4. Какие средства используются для увеличения дальности соединений в компьютерных сетях?
5. Какую роль играют сетевые службы? Назовите и охарактеризуйте известные вам сетевые службы.

Глава 22
Физическая и логическая структуризация сетей

В локальных сетях с небольшим количеством узлов обычно используются типовая топология (шина, кольцо, звезда) и однородные аппаратные средства. При увеличении масштаба сети с сохранением однородности ее структуры появляются ограничения на длину линий связи, на количество узлов в сети, а также на интенсивность трафика. Для снятия возникающих ограничений используются коммуникационные устройства и специальные методы структуризации сети.

ПРИМЕЧАНИЕ

Вообще говоря, коммуникациями называются возможные пути перемещения людей, грузов, сообщений и т. д. между двумя географически разнесенными точками. В компьютерных сетях коммуникационные устройства обеспечивают перемещение сообщений по линиям связи.

В качестве коммуникационных устройств используются повторители, ретрансляторы, концентраторы, мосты, коммутаторы, маршрутизаторы и шлюзы.

Обычно к одному и тому же сегменту электрического кабеля подключается до двух десятков компьютеров. В сетях, использующих соединения из электрических или оптоволоконных кабелей, для увеличения дальности соединений могут использоваться специальные устройства — **повторители**, или **репитеры** (от repeat — повторение), и **ретрансляторы**, связывающие отдельные сегменты кабелей и обеспечивающие за счет восстановления исходной мощности сигналов и фронтов возможность их дальнейшей передачи. Повторители и ретрансляторы снимают ограничения на дальность соединения, присущие одному кабельному сегменту (рис. 22.1).

Уже упоминавшийся ранее концентратор по сути дела является многовходовым повторителем. Отметим, что входы и выходы концентраторов, а также других коммуникационных устройств принято называть **портами**.

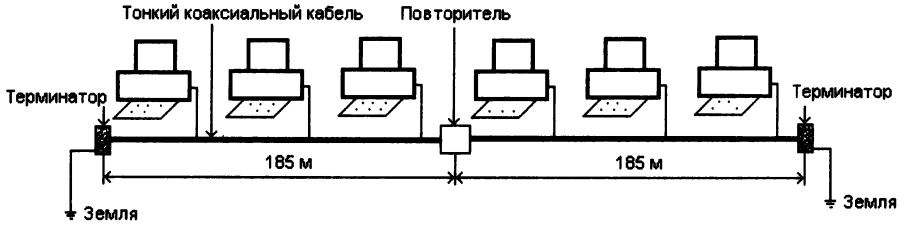


Рис. 22.1. Использование повторителя для увеличения дальности сетевых соединений

В общем случае концентраторы обеспечивают не только физическое соединение между несколькими компьютерами — с помощью внутренних настроек можно регулировать порядок передачи сообщений между подключенными к нему компьютерами. Например, в шинной топологии для организации широковещательной передачи концентратор дублирует входной сигнал на всех портах, а в кольцевой топологии — только на тот порт, к которому подключен следующий узел. Существуют и другие варианты работы концентраторов. Таким образом, с помощью концентраторов можно изменить физическую структуру, иначе говоря, топологию сети, оставляя без изменения ее логическую структуру, или, наоборот, изменить логическую структуру при неизменной или заданной физической.

Следует понимать, что физическая топология сети, или конфигурация физических связей между подключенными к сети компьютерами, определяется фактическими электрическими соединениями, в том числе повторителями и концентраторами. Логическая же структура сети определяется конфигурацией информационных потоков между компьютерами сети. При этом физическая топология и логическая структура сети могут не совпадать. На рис. 22.2 изображена сеть с физической топологией «звезда», в то время как реализуемые концентратором логические связи организованы в соответствии с топологией с общей шиной. Небольшая перенастройка концентратора дает возможность при неизменных физических соединениях создать кольцевую логическую структуру. Для этого достаточно организовать в концентраторе дополнительную цепочку передачи сообщений из порта C напрямую в порт A. Отметим, что физическая структуризация сети с помощью концентраторов может повысить *надежность* работы сети.

Анализ потоков сообщений, проходящих по различным участкам локальных сетей, выявил обычно существующую неоднородность трафика на разных участках сетей. Сеть фактически разбивается на некоторое количество устойчивых групп узлов, принадлежащих разным филиалам организации, ее отделам, рабочим группам специалистов и т. д. При этом наиболее интенсивный трафик наблюдается внутри групп, и только небольшая часть сообщений уходит за их рамки. Считается, что до 80 % трафика приходится на внутригрупповые сообщения и только 20 % на межгрупповые.

Если не учитывать этой особенности трафика локальной сети при ее построении, то неизбежны существенные задержки в передаче сообщений. Чтобы проиллюстрировать причину их возникновения, рассмотрим простой пример. На рис. 22.3 изображена схема локальной сети организации, состоящей из двух отделов, О1 и О2. Для этой сети выбрана топология с общей шиной, организованная с помощью логических связей в концентраторе. Пусть, например, сообщение передается с компьютера A на компьютер B, которые находятся в одном и том же отделе О1. На время передачи кадров этого сообщения шина оказывается заблокированной для любых других передач. Все компьютеры всех отделов будут ждать, пока не завершится передача. Будут задерживаться не только сообщения, передаваемые между отделами О1 и О2, но и сообщения, передаваемые внутри отдела О2. Например, будет задержана передача сообщения с компьютера D на компьютер E, которые находятся в отделе О2.

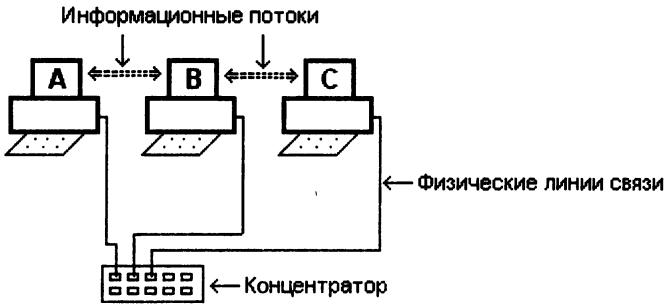


Рис. 22.2. Различие между физической топологией и логической структурой сети

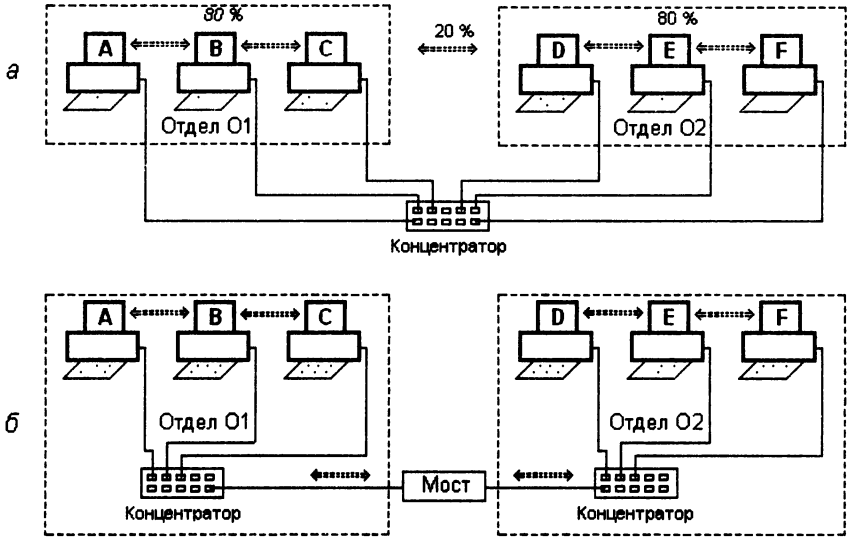


Рис. 22.3. Сеть: *а* — без логической структуризации; *б* — имеющая структуризацию

Чтобы повысить эффективность сети в целом, можно организовать логическую структуризацию сети, которая осуществляется с помощью разбиения сети на сегменты с локализованным в пределах группы трафиком. Распространение трафика, предназначенного для узлов некоторого сегмента сети, только внутри этого сегмента называется **локализацией трафика**. Схема построения сети с локализованным трафиком изображена на рис. 22.3, *б*. Компьютеры каждого из отделов образуют сегмент, который формируется с помощью отдельного концентратора. Связь между отделами организуется с помощью коммуникационного устройства, которое называется **мостом**. Мост представляет собой устройство, которое делит сеть на части — логические сегменты, осуществляя передачу кадра из одного сегмента в другой только при явной адресации к узлу из другого сегмента. Роль моста может играть специальная сетевая плата, устанавливаемая внутри компьютера, или же отдельный компьютер со специальным программным обеспечением.

Фактически, мост «не знает» топологии сети. Для локализации трафика он использует аппаратные адреса входящих в сеть компьютеров. При поступлении первого кадра от некоторого компьютера мост запоминает его аппаратный адрес и номер порта, на который поступил кадр. Во всех последующих передачах все поступающие по этому аппаратному адресу кадры мост выставляет на соответствующий ему порт, не выставляя их на остальные порты. Связывание аппаратного адреса компьютера и номера порта по первому поступившему от компьютера сообщению не обеспечивает возможности распознавания принадлежности данного узла к тому или иному сегменту сети.

Функции, аналогичные функциям моста, выполняет еще одна разновидность коммуникационных устройств — **коммутаторы**. По способу работы с кадрами они не отличаются от мостов. Однако кадры, поступающие от разных узлов, коммутаторы обрабатывают в параллельном режиме. Для этого каждый порт коммутатора имеет специализированный сопроцессор, который работает одновременно с сопроцессорами всех остальных портов.

Присущие мостам и коммутаторам ограничения по локализации трафика снимаются при использовании **маршрутизаторов**, или **роутеров** (от router — маршрут). Маршрутизаторы образуют логические сегменты с помощью явно заданной иерархии адресов, в которые включаются номера всех сегментов сети. Такой способ работы обеспечивает высокую эффективность и надежность ее работы. В качестве маршрутизаторов обычно используются мощные серверы.

Мосты и коммутаторы используются только для соединения сегментов локальных сетей с однотипным аппаратным оборудованием и программным обеспечением. Для соединения нескольких локальных сетей с различными типами аппаратного и программного обеспечения в каждой из сетей выделяется специализированный компьютер, который называется **шлюзом**. Соединенные друг с другом с помощью линий связи шлюзы каждой из сетей обеспечивают преобразование и пересылку информации от сети с одним методом ее передачи к сети с другим методом. Роль, которую в сетях играют шлюзы, существенно сложнее роли мостов.

Контрольные вопросы и упражнения

1. В чем проявляется различие между физической топологией и логической структурой сети?
2. Для чего организуется локализация трафика?
3. Сравните между собой функции и возможности концентраторов и мостов.
4. Сравните между собой функции и возможности мостов и коммутаторов.
5. Сравните между собой функции и возможности коммутаторов и маршрутизаторов.
6. Для чего используются шлюзы?

Глава 23
Доступ к сети

Только в сетях с полносвязной топологией для связи между двумя любыми узлами используются индивидуальные линии. В таких сетях любой из компьютеров может в любой момент времени передать сообщение любому другому компьютеру. При этом по различным линиям связи может одновременно передаваться любое количество сообщений. Во всех остальных топологиях для передачи сообщений несколько различных пар узлов совместно используют одну и ту же линию связи.

ВНИМАНИЕ

Линия связи, которая попеременно используется несколькими узлами для передачи сообщений, называется разделяемой. Вообще любой ресурс сети или компьютера, используемый не в монопольном режиме, называется разделяемым ресурсом. Главная цель использования разделяемых ресурсов — снижение общей стоимости системы.

Примером разделяемой линии связи является магистраль в сетях с топологиями типа «общая шина» или «кольцо» (см. рис. 21.1).

Из-за отсутствия прямых линий связи между рядом узлов сети и использования разделяемых линий возникает необходимость в коммутационном оборудовании, которое обеспечивает переключение разделяемой линии на обслуживание очередной пары взаимодействующих узлов. Узлы сети подключаются к коммутаторам по индивидуальным линиям связи, а коммутаторы связаны друг с другом разделяемыми линиями, которые используются всеми узлами совместно (рис. 23.1). Отметим, что такая схема соединений характерна и для обычных телефонных сетей.

Для разделяемых линий связи возникает несколько задач, обусловленных их попеременным использованием. Это, в частности, чисто электрические проблемы, связанные с согласованием электрических параметров соединений: сопротивления,

емкости и т. д. Однако основная группа проблем связана с отделением во времени доступа к разделяемой линии для различных пар узлов, обменивающихся сообщениями. Если, например, в сети с общей шиной несколько узлов начнут одновременно передавать свои сообщения на шину, то они взаимно уничтожат друг друга. Поэтому для передачи сообщения по сети любой из компьютеров должен:

- получить доступ к разделяемой линии связи;
- передать сообщение;
- освободить линию связи для других узлов, ожидающих своей очереди.

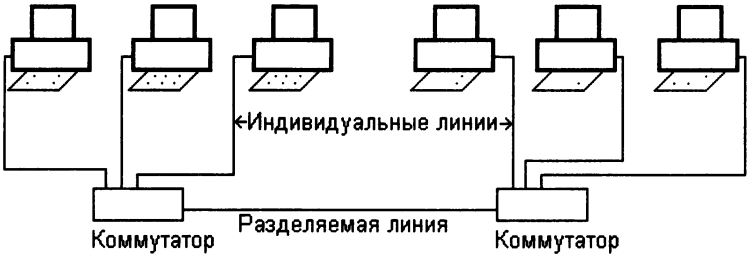


Рис. 23.1. Разделяемые линии связи

В локальных и глобальных сетях используется несколько различных методов получения доступа к разделяемым линиям, каждый из которых обладает определенными преимуществами и недостатками.

ВНИМАНИЕ

Правило, с помощью которого организуется доступ компьютеров к разделяемым линиям связи, принято называть методом доступа.

Основные методы доступа описываются в группе стандартов, которые разработаны в Институте инженеров по электротехнике и радиотехнике IEEE. Первоначально эта группа стандартов была предложена специалистами компаний Intel, DEC и Xerox, затем они были немного изменены IEEE и утверждены в качестве национального стандарта США. А чуть позднее Международная организация по стандартам **ISO** (от International Organization for Standardization) утвердила эту группу в качестве международного стандарта **ISO 8802**. Однако в литературе стандарт почти всегда употребляется под исходным названием **IEEE 802.x**, где **x** заменяется номером конкретного стандарта группы.

Стандарт IEEE 802.1 является общим документом, в котором вводятся основные понятия, описываются архитектура и общие процессы управления сетью. Стандарты IEEE 802.3, IEEE 802.4, IEEE 802.5, IEEE 802.6, IEEE 802.11, IEEE 802.12 и некоторые другие описывают характеристики и процедуры различных методов доступа к передающей среде.

23.1. Метод случайного доступа

В локальных сетях с общей шиной обычно используется описываемый стандартом IEEE 802.3 метод **множественного доступа с контролем несущей и обнаружением коллизий CSMA/CD** (от Carrier Sense Multiple Access/Collision Detection), который известен также как метод **случайного доступа**. Этот стандарт определяет несколько возможных структур передаваемого по шине кадра. В простейшем случае кадр имеет следующую структуру.

1. Преамбула, состоящая из 56 чередующихся нулей и единиц $010101...01_2$.
2. Однобайтовый начальный ограничитель кадра, содержащий фиксированную последовательность 10101011_2.
3. Сетевой адрес получателя, занимающий 6 байт.
4. Сетевой адрес отправителя, занимающий 6 байт.
5. Двухбайтовое поле длины кадра.
6. Тело кадра, представляющее собой передаваемое сообщение.
7. Контрольная сумма, занимающая 4 байта.

Хотя под длину кадра отведено 2 байта, его фактическая длина не может превышать 1500 байт. Ограничена также минимальная длина кадра, которая не может быть менее 64 байт. Если тело кадра имеет меньшую длину, то оно дополняется до минимальной специальным символом-заполнителем.

Важной частью метода является процедура проверки доступности линии связи для передачи. Компьютер, «желающий» передать сообщение, проверяет, свободна разделяемая линия или нет. Если во время проверки линия связи оказывается занятой, через определенное время компьютер-отправитель повторяет попытку начать передачу. После того как компьютер убедился в том, что линия связи свободна, он захватывает ее в монопольное использование и начинает передачу. Линия связи передается в монопольное владение узла только на время передачи одного кадра. После завершения его передачи линия связи освобождается.

В схеме метода случайного доступа предусматривается возможность ситуации, когда после обнаружения линии свободной два или более узла одновременно делают попытку начать передачу. Эту ситуацию принято называть **коллизией**. В случайном методе доступа при обнаружении коллизии все сетевые адаптеры, которые пытались передать свои кадры, прекращают передачу и после некоторой паузы случайной длительности снова пытаются получить доступ к линии связи. Узел, первым сделавший такую попытку, захватывает линию в монопольное использование, а остальные восстанавливают стандартный способ доступа.

Передающий компьютер осуществляет широковещательную передачу. Все узлы сети постоянно следят за состоянием магистрали (отсюда — *контроль несущей*), и как только на ней появляется передаваемое сообщение, все узлы одновременно начинают его прием. Анализируя адрес получателя кадра и сопоставляя его со своим собственным адресом, сетевой адаптер узла опознает направленное ему сообщение. Кадр записывается в буфер памяти компьютера только при совпадении сравниваемых адресов.

Процедуры согласования доступа к разделяемым линиям в общем случае могут потребовать много времени на «накладные» расходы. Для повышения производительности системы в целом могут применяться комбинированные подходы, сочетающие индивидуальные линии связи с несколькими коммутаторами, осуществляющими процедуры согласования.

23.2. Маркерный метод доступа

В локальных сетях с кольцевой топологией обычно используется *детерминированный* **маркерный** метод доступа, который описывается стандартом IEEE 802.5. В этом методе право на доступ к разделяемой линии связи циклически передается от одного узла сети к другому с помощью постоянно циркулирующего в кольце кадра специального формата, который принято называть **маркером** (от marker — условный знак). Кадр маркера может быть пустым или занятым. Узел, принявший пустой маркер, может передать с ним свое сообщение, переведя тем самым маркер в состояние занятого. Занятый маркер уже содержит переданное каким-то узлом сообщение. Узел, принявший занятый маркер, должен выяснить, кому адресовано сообщение, и если узел не является адресатом сообщения, передать кадр маркера в неизменном виде следующему узлу.

Кадр, воспринимающийся как пустой маркер, имеет следующую структуру.

1. Однобайтовый стартовый ограничитель, содержащий последовательность JK0JK000 манчестерского кода.
2. Байт информационного поля маркера, который, в частности, содержит равный единице бит T, показывающий, что маркер пуст.
3. Однобайтовый концевой ограничитель, содержащий комбинацию JK1JK1 манчестерского кода.

Кадр, соответствующий занятому маркеру, имеет следующую структуру.

1. Однобайтовый стартовый ограничитель, содержащий последовательность JK0JK000 манчестерского кода.
2. Байт информационного поля маркера, который, в частности, содержит равный нулю бит T, показывающий, что маркер не пуст.
3. Управляющий байт кадра, который используется для диагностики состояния кольца.
4. Сетевой адрес получателя, занимающий в зависимости от настроек 2 или 6 байт.
5. Сетевой адрес отправителя, занимающий в зависимости от настроек 2 или 6 байт.
6. Тело кадра, представляющее собой передаваемое сообщение.
7. Контрольная сумма, занимающая 4 байта.
8. Однобайтовый концевой ограничитель, содержащий комбинацию JK1JK1 манчестерского кода.
9. Байт состояния кадра, используемый для регистрации и опознания факта приема кадра получателем.

Рассмотрим немного подробнее действия узлов кольца с маркером. Передаваемый текущим узлом следующему маркер постоянно циркулирует по кольцу. Узел, которому нужно передать сообщение, дожидается очередного получения маркера и анализирует состояние бита T. Если маркер пуст ($T = 1$) и, следовательно, линия связи свободна, узел формирует кадр занятого маркера, в котором $T = 0$, включает в кадр передаваемое сообщение, а также адреса получателя и отправителя.

Каждый узел, получивший занятый маркер ($T = 0$), анализирует адрес получателя, и если он совпадает с собственным адресом узла, записывает полученный кадр в буфер компьютера. При этом в байт состояния передаваемого далее кадра включается признак, подтверждающий прием кадра адресатом. Если адрес текущего узла и адрес получателя не совпадают, кадр отправляется соседнему узлу в неизменном виде. Таким образом, пройдя по кольцу, кадр вернется к узлу-отправителю. Узел-отправитель, получив кадр, содержащий признак, который подтверждает его прием, изымает обработанное сообщение из линии связи и отправляет вместе с маркером следующее сообщение. Завершив передачу всех кадров сообщения, передающий узел возвращает маркеру исходное пустое состояние.

Чтобы избежать захвата маркера одним из узлов на неопределенное время, каждый узел получает его не более чем на 10 мс. По истечении этого времени маркер в любом случае должен быть освобожден узлом. Если узел за отведенное время не успевает передать все сообщение, то для передачи оставшихся кадров он снова должен дождаться получения пустого маркера.

Контрольные вопросы и упражнения

1. Что относится к разделяемым ресурсам? Для чего понадобилось вводить разделение ресурсов?
2. Какие действия должен выполнить компьютер, чтобы передать сообщение по компьютерной сети?
3. Что представляет собой метод доступа?
4. Опишите случайный метод доступа. Где он используется?
5. Опишите маркерный метод доступа. Где он используется?

Глава 24
Методы коммутации

После получения доступа к разделяемой линии связи узел должен передать сообщение адресату. В связи с тем, что к линии подсоединено несколько компьютеров, в сообщении необходимо определенным образом указать, какому именно компьютеру оно адресовано, а коммуникационное оборудование сети, включающее разделяемые линии связи, должно организовать его пересылку по указанному адресу.

В принципе, существует два типа технологий передачи сообщений: упоминавшаяся ранее широковещательная передача и передача от одного узла к другому узлу сети, которую принято называть **передачей от точки к точке**. Широковещательная сеть должна иметь единую среду передачи сообщений, к которой одновременно имеют доступ все ее узлы. Таким свойством обычно обладают локальные сети небольшого масштаба с топологиями типа «общая шина» и «кольцо». В этих сетях кадры сообщений, пересылаемые передающим узлом, принимаются всеми остальными узлами. Полученные узлом кадры, содержащие адрес, не совпадающий с собственным адресом узла, игнорируются. В случае совпадения адресов кадр принимается и копируется в буфер. Как частные случаи широковещательной передачи могут рассматриваться кадры, направляемые сразу всем узлам или некоторой группе узлов. Такая передача называется **многоадресной**. К широковещательным технологиям относятся используемые в локальных сетях способы передачи сообщений, входящие в обсуждавшиеся ранее методы случайного и маркерного доступа.

В глобальных сетях широко применяется передача от точки к точке. Такая сеть может рассматриваться как множество пар узлов, которые имеют отдельную линию связи друг с другом. Например, в сети, схема которой изображена на рис. 24.1, это множество состоит из пар {(A, B), (A, D), (B, C), (B, F), (C, D), (C, E), (D, F), (E, F)}. Кадр от узла-отправителя передается узлу-получателю по цепочке связанных друг с другом промежуточных узлов. Например, из узла A в узел F кадр может пройти по цепочке A–B–F. Таким образом, в каждой передаче участвует несколько пар узлов. При этом обычно существует несколько возможных

путей перемещения кадра от исходного узла к конечному. Выбор оптимального варианта является одной из основных задач коммутаторов и маршрутизаторов.

Конкретные способы передачи сообщений в сетях с передачей от узла к узлу принято называть **схемами коммутации**. В настоящее время существует три базовых схемы коммутации:

- коммутация каналов;
- коммутация пакетов;
- коммутация сообщений.

Вне зависимости от базовой схемы различают сети с **постоянной** коммутацией и сети с **динамической** коммутацией. В первом случае каждая пара узлов образует соединение на длительный срок, например на несколько месяцев. Соединение формируется обслуживающим персоналом сети путем соответствующей настройки оборудования. Постоянную коммутацию называют также режимом **выделенных**, или **арендуемых**, **каналов**. Во втором случае коммутация выполняется во время сеанса связи узлов по заказу одного из них, а затем также по инициативе одного из взаимодействующих узлов связь разрывается. Взаимодействие узлов при динамической коммутации может длиться от долей секунды до нескольких часов, в зависимости от базовой схемы коммутации.

24.1. Коммутация каналов

Сети с **коммутацией каналов** являются исторически первой формой коммутации. Эта форма появилась вместе с телефонными линиями. Коммутация каналов означает установление между узлами непрерывного составного физического канала из последовательно соединенных отдельных участков сети. До начала передачи данных необходимо выполнить процедуру установки соединения по типу соединения перед разговором в телефонной сети.

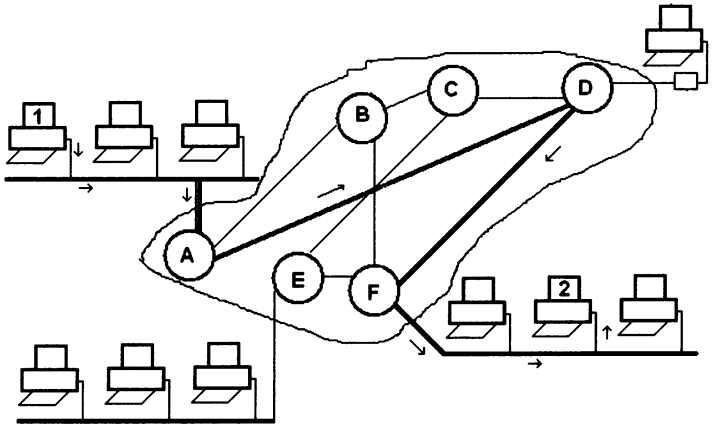


Рис. 24.1. Коммутация каналов

В качестве примера коммутации каналов рассмотрим ситуацию, изображенную на рис. 24.1. Узел 1 для связи с узлом 2 должен переслать запрос на установление соединения маршрутизатору (или коммутатору) A, указав адрес назначения. Маршрутизатор A должен выбрать дальнейший маршрут образования канала и передать запрос выбранному коммутатору и т. д. Запрос проходит до коммутатора F, связанного с узлом 2. Можно считать, что запрос как бы «прокладывает путь» от отправителя к получателю, по которому потом пойдет основное сообщение. Если узел 2 не занят и может принять запрос на соединение, он направляет по уже установленному каналу — выбранной цепочке маршрутизаторов — ответ исходному узлу. После этого канал считается скоммутированным, и передающий узел может начать передачу сообщения по фиксированной цепочке узлов и линий. Поскольку после установления соединения маршрут повторно выбирать не требуется, передача кадров сообщения выполняется с большей скоростью.

Во время пересылки основного сообщения все узлы и линии связи, входящие в такой «составной» канал, недоступны для других действий. После завершения передачи отправитель особым образом уведомляет получателя об окончании сеанса связи и инициирует разъединение, после чего «составной» канал прекращает функционирование, а узлы и линии связи освобождаются для других соединений.

В примере на рис. 24.1 «составной» канал проходит по маршруту A–D–F. Этот маршрут не является единственно возможным. Передача между рассматриваемыми узлами может быть осуществлена также по маршрутам A–B–F или A–B–C–E–F. Маршрутизаторы выбирают путь в соответствии с известными им характеристиками линий связи, такими как длина линий и их пропускные способности. При этом самый короткий маршрут не всегда является оптимальным. Если, например, пропускная способность участков D–F и B–F мала, то, скорее всего, будет выбран маршрут A–B–C–E–F. Для выбора пути передачи сообщений маршрутизаторы используют различные методы, например алгоритмы Дейкстры и Флойда для выбора кратчайшего пути в графе ([4], [31]).

Свойства соединения с помощью коммутации каналов:

- необходимость предварительной установки соединения, на которую может уйти довольно много времени;
- гарантированная высокая пропускная способность после установки соединения;
- неэффективность работы в условиях пульсирующего, то есть неравномерного, с большими паузами трафика, характерного для компьютерных сетей;
- невозможность использования аппаратуры, работающей с разной скоростью.

В связи с перечисленными особенностями схема с коммутацией каналов в настоящее время редко используется в компьютерных сетях.

24.2. Коммутация пакетов

В схеме с коммутацией каналов после установления соединения выбранный канал закрепляется за участвующими в обмене узлами на все время взаимодействия. При этом может оказаться, что некоторую (и даже значительную) часть времени

канал будет простаивать, причем остальные узлы в этот период не смогут установить соединение. Такая ситуация характерна для телефонных сетей, когда собеседники периодически обмениваются фразами с произвольными промежутками молчания между ними. При этом в периоды молчания занятая ими линия связи не может быть использована другими абонентами.

Характер занятости канала во время соединения описывается так называемым **коэффициентом пульсации**, который определяется как отношение средней интенсивности потока сообщений к максимально возможной. В компьютерных сетях коэффициент пульсации может доходить до 1:100. В этом случае применение базовой схемы с коммутацией каналов приведет к очень неэффективному использованию возможностей сети.

В качестве альтернативного способа передачи информации в конце 1960-х гг. были разработаны компьютерные сети с **коммутацией пакетов**. При коммутации пакетов сообщение, рассматриваемое как логически самостоятельная порция информации (например, запрос на передачу файла, передаваемый в ответе файл и т. д.), разбивается в исходном узле на небольшие порции, которые называют **кадрами** или **пакетами**. Каждый пакет содержит какую-либо часть сообщения, необходимую адресную информацию и его порядковый номер. Пакеты передаются по сети независимо друг от друга, возможно, даже по разным маршрутам, а после поступления в конечный пункт собираются в цельное сообщение. Такой способ передачи сообщений повышает общую эффективность использования сетевого оборудования, увеличивает общую пропускную способность сети. Но он требует использования в узлах сети высокоскоростных коммутаторов. Кроме того, в связи с задержками пакетов в промежуточных узлах на передачу сообщения уходит больше времени, чем в схеме с коммутацией каналов.

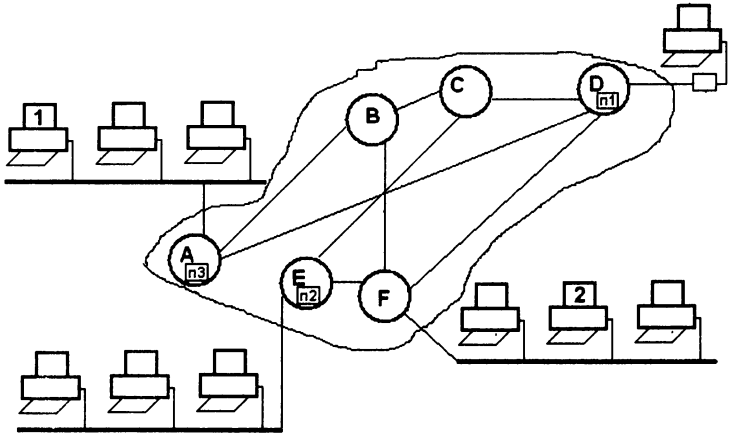


Рис. 24.2. Дейтаграммный режим коммутации пакетов

Маршрутизаторы или коммутаторы, реализующие пакетную коммутацию, имеют внутреннюю буферную память, в которой при необходимости накапливаются и временно хранятся транзитные пакеты. Это приводит к сглаживаниям

пульсации, которые, с одной стороны, могут вызывать задержки для отдельных узлов, а с другой — увеличивают общую эффективность использования коммуникационного оборудования.

Коммутация пакетов может происходить в **дейтаграммном** режиме или в режиме виртуальных каналов. При работе в дейтаграммном режиме маршруты разных пакетов могут различаться. На рис. 24.2 изображена ситуация, в которой пакет п1, находящийся в точке D маршрута, и пакет п2, обрабатываемый маршрутизатором E, проходят из узла 1 в узел 2 разными маршрутами. Пакет п1 проходит маршрут A–D–F (или A–B–C–D–F), в то время как пакет п2 проходит по маршруту A–B–C–E–F. При работе в режиме виртуальных каналов все пакеты следуют по одной и той же цепочке коммутаторов, которые таким образом как бы образуют скоммутированный виртуальный канал.

24.3. Коммутация сообщений

Сети с коммутацией сообщений появились в ходе разработки сетей с коммутацией пакетов. При использовании этой разновидности коммутации все сообщение целиком передается между транзитными узлами сети с временной буферизацией (записью на диск) на каждом транзитном компьютере, через который оно проходит. Это может быть электронное письмо, файл с документом или программой, передаваемые в автономном, или **оффлайновом** (от off line — вне линии), режиме. Транзитные компьютеры при этом могут работать как в режиме коммутации каналов, так и в режиме коммутации пакетов. В настоящее время коммутация сообщений в чистом виде практически не используется.

Контрольные вопросы и упражнения

1. Чем отличается широковещательная передача от передачи от точки к точке?
2. Назовите и кратко охарактеризуйте базовые коммутационные схемы.
3. Почему в компьютерных сетях неэффективно использование коммутации каналов?
4. Сравните между собой дейтаграммный режим и режим виртуальных каналов.

Глава 25
Базовые сетевые технологии

В каждом конкретном случае развертывания компьютерной сети в той или иной организации разработчики не начинают свою работу на голом месте, заново проектируя все компоненты сети, от выбора кабеля до выбора и настройки программного обеспечения. Обычно они выбирают наиболее подходящую в конкретной ситуации **сетевую технологию**, применяемую во многих аналогичных случаях.

ВНИМАНИЕ

Сетевой технологией называется согласованный набор стандартных соглашений и правил и реализующий их комплекс программно-аппаратных средств (драйверов, адаптеров, кабелей и т. д.), достаточный для построения работоспособной сети.

В настоящее время для построения локальных сетей в основном применяются базовые сетевые технологии: Ethernet, Token Ring, FDDI, 100VG-Any LAN, Fast Ethernet, Gigabit Ethernet, Wi-Fi, а также некоторые другие.

Стандарт IEEE 802.3 сетевой технологии **Ethernet** (эфирная сеть) был принят в 1980 г. По некоторым данным, количество сетей, построенных по этой технологии, в настоящее время превышает 5 миллионов. Сети Ethernet используют топологию с общей шиной и случайный (CSMA/CD) метод доступа к разделяемой линии связи. Доступ к шине осуществляется через сетевые адаптеры компьютеров. Каждый адаптер получает уникальный аппаратный номер. В качестве шины может использоваться витая пара, тонкий или толстый коаксиальный либо оптоволоконный кабель. Передача данных осуществляется со скоростью 10 Мбит/с. Максимальное удаление узлов в сети составляет 2,5 км, а общее количество компьютеров не более 1024. Основными достоинствами сети Ethernet являются экономичность, легкая масштабируемость и простота обслуживания.

Стандарт IEEE 802.5 сетевой технологии **Token Ring** (маркерное кольцо) был разработан компанией IBM в 1984 г. Сети Token Ring используют кольцевую топологию и маркерный метод доступа. Так же, как в сети Ethernet, в качестве

магистрали могут использоваться витая пара, тонкий или толстый коаксиальный либо оптоволоконный кабель. Пропускная способность магистрали до 16 Мбит/с. Максимальная длина кольца — 4 км, но количество компьютеров не может превышать 260, что значительно меньше, чем в технологии Ethernet. Однако сеть Token Ring обладает повышенной отказоустойчивостью по сравнению с сетью Ethernet.

Стандарт сетевой технологии **FDDI** (от Fiber Distributed Data Interface — оптоволоконный интерфейс распределенных данных) был разработан в период с 1986 по 1988 г. Сетевая технология FDDI базируется на технологии Token Ring. В качестве магистрали предусмотрено использование двойной волоконно-оптической линии. Топология сети FDDI кольцевая, метод доступа — маркерный. Пропускная способность до 100 Мбит/с. Максимальная длина кольца 100 км, а максимальное количество компьютеров — 500. FDDI считается наиболее отказоустойчивой технологией локальных сетей.

Развитием технологии Ethernet считается сетевая технологии Fast Ethernet (быстрая эфирная сеть), описанная в принятом в 1995 г. стандарте IEEE 802.3a. От Ethernet унаследованы топология с общей шиной и случайный метод доступа. В качестве магистрали предусмотрено использование сдвоенной или счетверенной витой пары или оптоволоконного кабеля, что обеспечивает увеличение пропускной способности до 100 Мбит/с.

Современным вариантом сетевой технологии Ethernet является технология **Gigabit Ethernet** (гигабитная эфирная сеть), описанная в принятом в 1999 г. стандарте IEEE 802.3z. Как и в предыдущих вариантах Ethernet, используются топология с общей шиной и случайный метод доступа. Предусмотрено использование в качестве магистрали счетверенной витой пары или оптоволоконного кабеля, которые обеспечивают увеличение пропускной способности до 1 Гбит/с.

Стандарт IEEE 802.12 сетевой технологии 100VG-Any LAN был принят в 1995 г. В этом стандарте предусмотрена поддержка возможности работы с кадрами форматов Ethernet и Token Ring (отсюда Any LAN — любая локальная сеть). Сети Any LAN имеют древовидную топологию. В них используется метод доступа с названием **протокол приоритетных запросов**, **DPP** (от Demand Priority Protocol). В этом методе, в отличие от метода случайного доступа, исключается возможность возникновения коллизий. Пропускная способность этих сетей доходит до 100 Мбит/с.

Стандарт IEEE 802.11, больше известный под названием **Wi-Fi** (от Wireless Fidelity — беспроводная безукоризненная точность), описывает беспроводную связь по радиоканалу. Существует три разновидности этого стандарта, IEEE 802.11a, IEEE 802.11b и IEEE 802.11g, которые обеспечивают доступ на относительно небольших расстояниях, но с довольно большой скоростью.

Для передачи данных по стандарту IEEE 802.11b используется диапазон частот от 2,4 до 2,4835 ГГц, дальность связи не превышает 100 м, а максимальная скорость передачи равна 11 Мбит/с. Два других стандарта, IEEE 802.11a и IEEE 802.11g, работают на частотах 5 и 2,4 ГГц соответственно при максимальной скорости передачи данных 54 Мбит/с. Однако эти технологии обеспечивают меньший

радиус распространения сигналов и требуют гораздо более мощных процессоров для обработки и кодирования информации.

Ядром беспроводной сети является точка доступа (рис. 25.1). Вокруг нее образуется территория радиусом 50–100 м, называемая **хот-спотом** (от hot spot — горячее пятно), или **зоной Wi-Fi**. Любой компьютер с сетевым адаптером, поддерживающим технологию Wi-Fi, попав в эту зону, сразу же оказывается подключенным к сети, так как все необходимые настройки производятся автоматически. После этого работа пользователя осуществляется точно так же, как в любой проводной локальной сети.

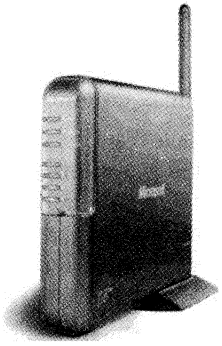


Рис. 25.1. Точка доступа беспроводной сети (фотография с сайта ej.typepod.com)

В настоящее время уже существует множество устройств, поддерживающих стандарт Wi-Fi. Это ноутбуки, объединенные с точкой доступа ADSL-модемы, беспроводные MP3-плееры, принтеры и т. д.

Кроме стандарта Wi-Fi в настоящее время предложено еще несколько беспроводных стандартов. Это, например, группа стандартов сотовой связи, стандарт передачи на большие расстояния **WiMax** (от Wireless Maximum) и стандарт **Bluetooth** (синий зуб), который описывает объединение вычислительных устройств, например карманных компьютеров, и средств сотовой связи, например сотовых телефонов.

Контрольные вопросы и упражнения

1. Что называется базовой сетевой технологией?
2. Охарактеризуйте сетевую технологию Ethernet.
3. Охарактеризуйте сетевую технологию Token Ring.
4. Охарактеризуйте сетевую технологию FDDI.
5. Охарактеризуйте сетевую технологию Gigabit Ethernet.
6. Охарактеризуйте сетевую технологию 100 VG Any-LAN.
7. Охарактеризуйте сетевую технологию Wi-Fi.

Глава 26
Многоуровневая модель OSI

Организация взаимодействия различных узлов в сети является сложной задачей. Для ее решения используется универсальный прием *декомпозиции*, суть которого состоит в сведении сложной задачи к последовательности из нескольких более простых задач. В рамках этого подхода программное и аппаратное обеспечение каждого узла сети рассматривается как система, состоящая из нескольких иерархически связанных уровней, каждому из которых приписывается определенный набор сетевых функций и обязанностей. Обсуждаемый способ организации работы узла сети называется **многоуровневым подходом**. Вообще говоря, многоуровневый подход используется довольно широко — например, файловые подсистемы операционных систем являются многоуровневыми.

Программные и аппаратные модули, образующие некоторый уровень системы, формируются таким образом, чтобы они обращались только к модулям нижележащих уровней, а результаты их работы передавались только вышележащим уровням. На каждом уровне системы фиксируется полный перечень его функций, а также межуровневые интерфейсы, обеспечивающие передачу данных с одного уровня на другой.

Напомним, что интерфейсом называется стандартизированный и унифицированный набор аппаратных и программных средств, а также набор правил взаимодействия устройств или программ друг с другом и с пользователем, в том числе набор правил передачи информации по линиям связи. Обычно интерфейс реализуется с помощью контроллеров и драйверов внешних устройств компьютера. Характерными примерами используемых в компьютерах интерфейсов являются параллельный интерфейс, используемый при передаче информации на принтер, и последовательный интерфейс, используемый для обмена с мышью, модемом и т. д.

Перечисленные ранее соглашения о формировании уровней системы являются, по сути дела, базовыми принципами многоуровневого подхода. Они обеспечивают относительную независимость любого уровня системы и возможность такой замены любого уровня, которая не затрагивает все остальные ее уровни.

Для описания работы отдельно взятого компьютера достаточно понятия интерфейса. Если требуется описать совместную работу по обмену данными для нескольких машин, играющих роль узлов компьютерной сети, то ситуация усложняется и одного понятия интерфейса оказывается недостаточно. В этом случае многоуровневый подход имеет специфические особенности. Они обусловлены тем, что при взаимодействии любых двух компьютеров сети нужно обеспечить согласованную работу по крайней мере двух многоуровневых иерархических систем: одной на передающей стороне, а другой — на принимающей.

Процедура взаимодействия двух узлов сети может быть описана некоторой совокупностью правил. Эта совокупность, так же как и аппаратное и программное обеспечение узлов, может быть разбита на иерархические уровни. Для каждой пары соответствующих друг другу (имеющих, например, один и тот же номер), но находящихся в разных узлах уровней можно выделить свой набор правил, которые образуют **протоколуровня**.

ВНИМАНИЕ

Формализованная система правил, определяющих последовательность и формат сообщений, которыми обмениваются сетевые компоненты, лежащие на одном уровне, но в разных узлах, называется протоколом уровня.

Как следует из изложенного ранее, модули, реализующие протоколы двух соседних уровней одного и того же узла, также взаимодействуют друг с другом по четко определенным правилам, которые образуют интерфейс (рис. 26.1).

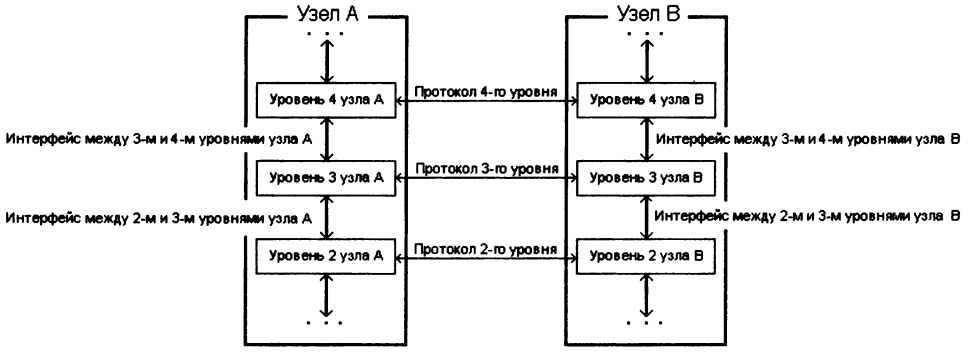


Рис. 26.1. Интерфейсы и протоколы многоуровневого взаимодействия узлов в сети

ВНИМАНИЕ

Модули каждого уровня отрабатывают протоколы своего уровня и интерфейсы с соседними уровнями.

По сути, интерфейс и протокол представляют собой одно и то же — это некоторая система правил взаимодействия, но применяются эти системы правил к разным объектам. Интерфейсы связывают уровни одного узла, находящиеся в отношении

старшинства и подчиненности, а протоколы связывают равноправные уровни разных узлов. Чтобы отразить это различие, использованы разные термины.

Для полного описания взаимодействия двух узлов сети каждый из уровней узлов должен быть описан отдельным протоколом. Протоколы всех уровней образуют иерархическую систему, организованную в **стек** протоколов.

ВНИМАНИЕ

Иерархически организованный набор протоколов, достаточный для организации взаимодействия узлов в сети, называется стеком коммуникационных протоколов.

В начале 1980-х гг. международная организация ISO и ряд других организаций разработали совокупность стандартов, описывающих функционирование и возможности сетей различных масштабов. Эту совокупность стандартов назвали **эталонной моделью взаимодействия открытых систем OSI** (от Open System Interconnection). Полное описание модели OSI представляет собой документ, содержащий более тысячи страниц.

Следует подчеркнуть, что в широком смысле открытой системой считается любая система с открытыми, то есть опубликованными стандартами, содержащими исчерпывающее формализованное[1] описание какого-либо объекта. В узком смысле, который обычно приписывается термину «открытая» в названии OSI, это понятие воспринимается как сетевая структура, доступная для подключения дополнительных сетей с любой внутренней структурой.

ВНИМАНИЕ

В настоящее время модель OSI рассматривается как эталонное теоретическое описание, удобное для обсуждения общих принципов построения сетей, выработки общих подходов и рекомендаций, а не как реально используемая совокупность сетевых программных и аппаратных средств и стандартов.

Модель OSI определяет семь уровней взаимодействия систем в сетях, а также исчерпывающим образом описывает функции каждого уровня и межуровневые интерфейсы. Поскольку в модели OSI все средства сетевого взаимодействия организованы в семь иерархических уровней, ее часто называют **семиуровневой** моделью. Модель OSI содержит описания физического, канального, сетевого, транспортного, сеансового, представительного (в смысле способа представления данных) и прикладного уровней.

Три нижних уровня, физический, канальный и сетевой, относятся к **сетезависимым**, они тесно связаны с технической реализацией сети и с используемым коммуникационным оборудованием. Три верхних уровня, сеансовый, представительный и прикладной, ориентированы на приложения и практически не зависят от технической реализации. Транспортный уровень является промежуточным,

[1] Формализованным называется описание, записанное по определенным правилам, которые позволяют избежать неоднозначности его трактовки.

он служит для отделения этих групп уровней друг от друга. Выполняющаяся программа, в принципе, может взять на себя функции верхних трех уровней (прикладного, представительного и сеансового), но она не вправе вторгаться на более низкие уровни.

Кратко ознакомимся с принятой в модели OSI общей схемой взаимодействия двух многоуровневых узлов. Пусть, например, выполняющееся на одном компьютере приложение запрашивает некоторые данные, находящиеся на другом компьютере сети. Для этого приложение должно по правилам, описанным в интерфейсе прикладного уровня, сформулировать запрос, который обычно адресуется к файловой службе другого компьютера. Запрос имеет вид сообщения, которое состоит из поля данных, заголовка и, возможно, хвостовой части (рис. 26.2).

Рис. 26.2. Общая структура сообщения

Заголовок содержит необходимую служебную информацию, например имя файла, тип операции (чтение, запись), которую в соответствии с протоколом прикладного уровня нужно по сети передать прикладному уровню приложения адресата. В общем случае поле данных может быть пустым или содержать какую-либо информацию. Хвостовая часть может содержать, например, контрольные суммы, обеспечивающие обнаружение или исправление ошибок.

Этот общий принцип взаимодействия соседних уровней конкретизируется в семиуровневой модели OSI следующим образом. К полученному от приложения сообщению прикладной уровень добавляет заголовок h1 и, возможно, хвостовую часть t1, содержание которых определяется протоколом этого уровня. Далее эта информация оформляется в соответствии с правилами межуровневого интерфейса и направляется представительному уровню (рис. 26.3). На нем повторяются те же самые действия — в соответствии с протоколом представительного уровня формируются заголовок второго уровня h2 и, возможно, хвостовая часть второго уровня t2, затем вся информация оформляется по правилам следующего межуровневого интерфейса и направляется сеансовому уровню. Аналогичные действия производятся на каждом уровне. После последнего, седьмого уровня к исходному сообщению в зависимости от требований промежуточных уровней может быть подключено до семи заголовочных и семи хвостовых частей. Последний уровень направляет всю передаваемую информацию в линию связи для побитовой передачи по сети.

Во время подъема по уровням на машине-адресате каждый очередной уровень обрабатывает заголовок и хвостовую часть своего уровня, выполняет описанные в них действия, убирает обработанные заголовок и хвостовую часть и передает оставшуюся часть дальше. В конце пути приложение-адресат получает только исходное сообщение. Все промежуточные заголовки и хвостовые части разных уровней обеспечивают его безошибочную передачу и правильное воспроизведение в условиях другого приложения, которые могут существенно отличаться от условий в передающем приложении.

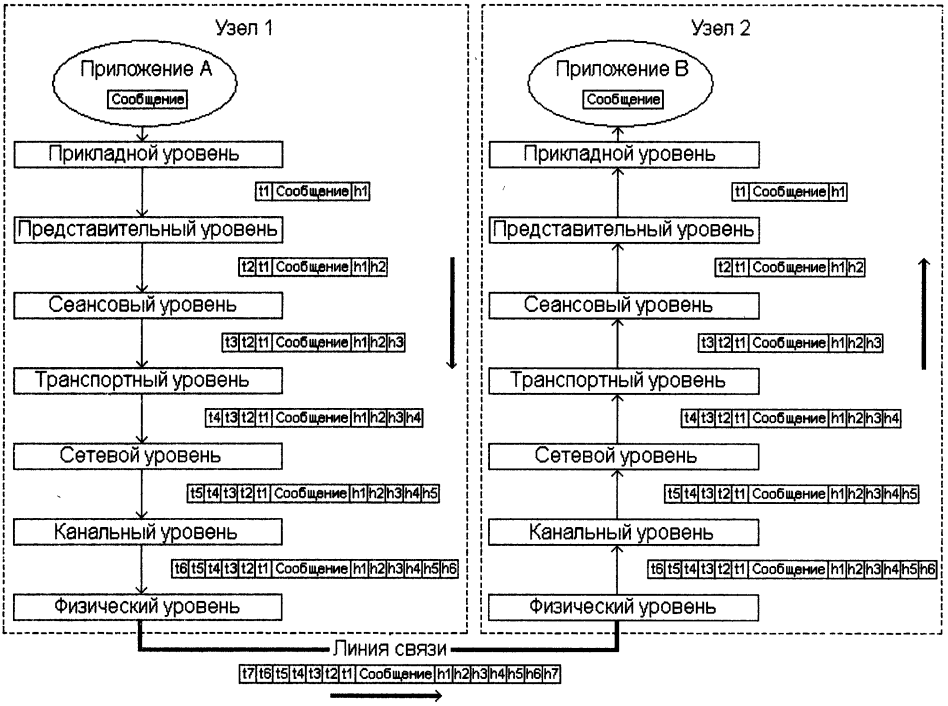


Рис. 26.3. Схема передачи сообщения между узлами в модели OSI

В стандарте OSI для обозначения единиц данных, с которыми имеют дело протоколы различных уровней, используется общее название **протокольный блок данных**, или **PDU** (от Protocol Data Unit). Для обозначения блоков данных конкретных уровней могут использоваться специальные названия: **кадр**, **пакет**, **дейтаграмма**, **сегмент**.

В стандарте OSI определены только две разновидности протоколов коммутации. Это протоколы с установленным соединением, то есть протоколы коммутации каналов, и дейтаграммные протоколы без предварительной установки соединения, то есть протоколы коммутации пакетов.

На *физическом уровне* модели OSI описывается процесс передачи битов по физической линии связи. Протокольным блоком данных является бит. На этом уровне предусмотрены стандарты экранированных и неэкранированных витых пар, тонких и толстых коаксиальных кабелей, оптоволоконных кабелей, радиоканалов, описываются их физические характеристики, такие как волновое сопротивление, полоса пропускания, крутизна фронта импульса, скорость передачи данных, стандарты и назначение каждого контакта используемых разъемов. Например, упоминавшаяся ранее спецификация 10Base-T представляет собой типичный пример стандарта физического уровня.

На *канальном уровне* модели OSI отрабатываются приемы совместного использования разделенных линий связи. Протокольным блоком данных считается кадр.

Протоколы канального уровня содержат описания алгоритмов проверки доступности линии связи (например, протокол CSMA/CD), алгоритмов обнаружения и коррекции ошибок и т. д. Канальный уровень обеспечивает корректную передачу каждого кадра с помощью размещения в каждом кадре специальных контрольных сумм. При обнаружении ошибки передачи кадра канальный уровень может повторно передать поврежденный кадр. Однако исправление ошибок не является обязанностью протоколов канального уровня. Канальный уровень обеспечивает передачу кадра между любыми двумя узлами сети стандартной топологии (шина, кольцо, звезда). В компьютере функции канального уровня возложены на сетевые адаптеры и драйверы операционных систем. Протоколы канального уровня исполняются в узлах локальных сетей концентраторами, мостами, коммутаторами и маршрутизаторами. В глобальных сетях канальный уровень обеспечивает обмен кадрами только для узлов, связанных индивидуальными линиями связи. Для этого используются протоколы с названием **PPP** (от Protocol Point to Point).

На *сетевом уровне* OSI уточняется смысл термина «сеть». Сетью считается совокупность компьютеров, объединенных в соответствии со стандартной сетевой технологией и использующих для обмена сообщениями протоколы канального уровня. А объединение сетей с различной технологией для обмена сообщениями использует уже протоколы сетевого уровня. Другими словами, сетевой уровень модели OSI служит для образования из нескольких сетей с разными топологиями и разными принципами передачи сообщений между узлами единой среды обмена сообщениями, например для объединения сетей с технологиями Ethernet и Token Ring. Сети соединяются между собой с помощью маршрутизаторов, то есть устройств, собирающих информацию о топологии межсетевых связей. Протокольным блоком данных является пакет. Для доставки пакетов на сетевом уровне используется понятие **номер сети**, и адрес получателя состоит по крайней мере из двух частей: номера сети и номера узла в сети.

Для передачи пакета от отправителя, находящегося в одной сети, к получателю, находящемуся в другой, нужно совершить несколько транзитных передач по наиболее подходящим маршрутам, которые и выбираются маршрутизаторами. Еще раз подчеркнем, что кратчайший путь между узлами в разных сетях не всегда оптимален. Главная задача сетевого уровня состоит в подборе оптимальных маршрутов пересылки пакетов по сети.

Как уже отмечалось, *транспортный уровень* модели OSI является связующим звеном между тремя нижними (физическим, канальным и сетевым) и тремя верхними (сеансовым, представительным и прикладным) уровнями или выполняющими их функции приложениями (программами). Он обеспечивает передачу данных верхним уровням с высокой степенью надежности.

Модель OSI определяет для транспортного уровня пять классов передачи данных, отличающихся друг от друга срочностью, возможностью восстановления прерванной связи, способностью к обнаружению и исправлению ошибок передачи данных. Выбор класса транспортного уровня зависит, с одной стороны, от степени надежности передачи данных на нижних уровнях, с другой — от требований

приложения или более высоких уровней. Если, например, качество линий связи на физическом и канальном уровнях высокое, то можно отключить многочисленные проверки, выбрав облегченный класс транспортного уровня. И наоборот, при низком качестве физического и канального уровней приходится выбирать наиболее мощный класс транспортного уровня, подключающий максимально возможный набор средств обнаружения и исправления ошибок передачи данных.

Протоколы *сеансового уровня* модели OSI обеспечивают фиксацию передающей и принимающей сторон, синхронизацию их взаимодействия, управление диалогом, выставление контрольных точек, позволяющих при сбое возвращаться к контрольной точке, а не к началу передачи. Сеансовый уровень часто объединяется с функциями более высоких уровней.

Протоколы представительного уровня OSI описывают форму представления и синтаксис передаваемой по сети информации для ее передачи на прикладной уровень или в приложение узла адресата. Фактически эти протоколы определяют способы преобразования информации между различными системами кодировок, использованными при передаче информации. Кроме того, они обеспечивают необходимый уровень секретности. Протоколы этого уровня играют существенную роль в обеспечении собственно «открытости» сетевых систем с самими разными способами внутреннего представления данных.

С помощью протоколов *прикладного уровня* пользователь получает доступ к различным сетевым ресурсам. К этому уровню относятся протокол службы передачи файлов **FTP** (от File Transfer Protocol), основной протокол Всемирной паутины **HTTP** (от Hyper Text Transfer Protocol — протокол передачи гипертекстов), протокол службы удаленного сетевого доступа **Telnet** (от TErminaL over NETwork protocol), протоколы служб электронной почты **POP** (от Post Office Protocol — протокол почтового отделения) и **SMTP** (от Simple Mail Transfer Protocol — простой протокол передачи почты) и многие другие протоколы.

Большинство реальных стеков коммуникационных протоколов на нижних, сетезависимых уровнях используют одни и те же стандартизированные протоколы Ethernet, Token Ring и т. д. Различия в основном проявляются на верхних уровнях. Протоколы этих уровней в различных стеках часто не соответствуют рекомендуемому OSI разбиению. Обычно все три верхних уровня объединяют в один.

Заметим, что эталонный стек протоколов OSI представляет собой международный независимый от производителей стандарт. Он отличается высокой сложностью, которая требует больших затрат вычислительных мощностей. Поэтому стек OSI на практике не используется.

В настоящее время в реально работающих сетях используются несколько различных стеков коммуникационных протоколов. К наиболее известным относятся стеки TCP/IP и IPX/SPX.

Стек протоколов TCP/IP был разработан несколько десятилетий назад при создании сети ARPAnet, родоначальницы Интернета. Сегодня этот стек протоколов считается одним из самых популярных и распространенных протоколов в мире. Основными протоколами стека являются сетевой протокол IP (от Internet protocol)

и транспортный TCP (от Transmitting control protocol). Протоколы IP обеспечивают продвижение пакета по сети, поддерживая все популярные стандарты физического и канального уровней, в том числи стандарты Ethernet, Token Ring, FDDI и т. д., а протоколы уровня TCP обеспечивают надежность передачи. За время эксплуатации стек вобрал в себя большое количество протоколов прикладного уровня, например уже упоминавшиеся протокол пересылки файлов FTP, почтовый протокол SMTP, протокол WWW и т. д.

Стек протоколов IPX/SPX разработан фирмой Novell для сетевой операционной системы NetWare. Основой стека являются протокол сетевого уровня IPX (от Internet work Packet exchange) и протокол транспортного уровня SPX (от Sequenced Packed exchange). Стек IPX/SPX распространен гораздо меньше, чем TCP/IP, так как операционная система Novell NetWare менее популярна, чем операционная система Windows.

Дополнительный материал по обсуждавшимся в третьей части вопросам можно найти в изданиях [26], [31], [32] [16], [15].

Контрольные вопросы и упражнения

1. Какие соглашения лежат в основе формирования отдельных уровней модели OSI?
2. Сравните между собой понятия «интерфейс» и «протокол».
3. Опишите общую схему передачи сообщений между двумя узлами в модели OSI.
4. Поясните смысл терминов «протокольный блок данных», «кадр», «пакет», «дейтаграмма». Сравните их между собой.
5. Охарактеризуйте назначение протоколов физического уровня.
6. Охарактеризуйте назначение протоколов канального уровня.
7. Охарактеризуйте назначение протоколов сетевого уровня. Сравните между собой протоколы этих уровней.
8. Охарактеризуйте назначение протоколов транспортного уровня.
9. Охарактеризуйте содержание сеансового, представительного и прикладного уровней.
10. Что представляет собой стек коммуникационных протоколов?
11. Охарактеризуйте стек протоколов TCP/IP.

Приложение
Использованные сокращения

ABC, Atanasoff – Berry Computer – название компьютера.

ADSL, Asymmetric Digital Subscriber Line – асимметричная цифровая абонентская линия.

AGP, Advanced Graphic Port – улучшенный графический порт, тип шины.

AMD, Advanced Micro Devices (улучшенные микроустройства) – название фирмы-производителя процессоров.

ASCI White, Accelerated Strategic Computing Initiative (ускоренная стратегическая вычислительная инициатива) – модель суперкомпьютера.

ASCII, American Standard Code for Information Interchange (американский стандартный код для обмена информацией) – распространенная кодовая таблица.

AT, Advanced Technology (продвинутая технология) – стандарт системного блока.

ATX, AT extended (улучшенный AT) – стандарт системного блока.

AVI, Audio-Video Interleaved (чередующиеся аудио и видео) – формат видео.

BCD, Binary Coded Decimal (двоично-кодированный десятичный формат) – числовой формат.

BEDO DRAM, Burst EDO DRAM (расширенное время удержания данных на выходе с блочным доступом) – тип микросхемы памяти.

BIOS, Base Input/Output system (базовая система ввода/вывода) – часть операционной системы.

BSB, Back Side Bus – шина, связывающая процессор с внешним кэшем.

CAS#, Column Address Strobe (строб адреса столбца) – управляющий сигнал микросхемы памяти.

CC NUMA, Coherent Cache NUMA (системы с согласованной кэш-памятью) – класс вычислительных систем.

CD, Compact Disc – компактный диск.

CD-R, Compact Disk Recordable (записываемый компакт-диск) — тип оптического диска.

CD-ROM, Compact Disk Read Only Memory — память только для чтения на компакт-дисках.

CD-RW, Compact Disk Rewriteable (перезаписываемый компакт-диск) — тип оптического диска.

CGA, Colour Graphics Adapter — цветной графический адаптер.

CISC, Compete Instruction Set Computer или **Code** (компьютер с полным набором инструкций (машинных команд) или полный набор кодов команд) — тип архитектуры системы команд.

CMOS RTC, CMOS Real Time Clock (память часов реального времени) — тип памяти компьютера.

CMOS, Complimentary Metal Oxide Semiconductor (комплементарный металл-оксид-полупроводник) — тип памяти компьютера.

COMA, Cache Only Memory Access (системы с доступом только к кэш-памяти) — класс вычислительных систем.

COW, Cluster Of Workstation (кластер рабочих станций) — класс вычислительных систем.

CPU, Central Processing Unit — центральный обрабатывающий блок, устройство.

CRC, Cyclic Redundancy Check — циклический избыточный контрольный код.

CRT, Cathode Ray Terminal (терминал на катодно-лучевой трубке) — тип дисплея.

CS, Chip Select (выбор чипа, микросхемы) — управляющий сигнал микросхемы памяти.

CSMA/CD, Carrier Sense Multiple Access/Collision Detection — метод множественного доступа с контролем несущей и обнаружением коллизий.

DBC, database computer (машина баз данных) — класс вычислительных систем.

DDCD, Double Density CD (компакт-диск двойной плотности) — тип оптического диска.

DDR SDRAM, Double Data Rate SDRAM (синхронизированная динамическая память с удвоением данных) — тип микросхемы памяти.

DEC, Digital Equipment Corporation (Корпорация цифрового оборудования) — название фирмы-производителя компьютеров.

DFC, Data Flow Computers (компьютер потоковых данных) — класс архитектуры вычислительных систем.

DIB, Dual Independent Bus (двойная независимая шина) — архитектура компьютера.

DIMM, Dual In Line Memory Modules — двухрядные модули памяти.

DMA, Direct Memory Access (прямой доступ к памяти) — название контроллера.

dpi, dots per inch (точки на дюйм) — разрешение при печати, сканировании.

DPL, Descriptor Privilege Level (код уровня привилегий) — часть дескриптора сегмента памяти.

DPP, Demand Priority Protocol — протокол приоритетных запросов.

DRAM, Dynamic Random Access Memory — динамическая память с произвольным доступом.

DVD, Digital Versatile Disk (цифровой многосторонний универсальный диск) — тип оптического диска.

EARN, European Academic and Research Network — Европейская академическая исследовательская сеть.

ECC, Error Checking and Correcting (контроль и исправление ошибок) — аппаратный механизм контроля оперативной памяти.

ED, Expand Down (расширение вниз) — часть дескриптора сегмента памяти.

EDO DRAM, Extended Data Out DRAM (расширенное время удержания данных на выходе) — тип микросхемы памяти.

EDSAC, Electronic Delay Storage Automatic Calculator (автоматический вычислитель с электронной памятью на линиях задержки) — название компьютера.

EDVAC, Electronic Discrete Automatic Variable Computer (параметризируемый автоматический электронный вычислитель дискретного действия) — название компьютера.

EEPROM, Electrically Erasable Programmable ROM (электрически стираемые перепрограммируемые ПЗУ) — тип памяти компьютера.

EGA, Enhanced Graphics Adapter — улучшенный графический адаптер.

EIDE, Extended IDE (расширенная IDE) — тип шины.

EIDE, Extended IDE (усовершенствованный IDE) — стандарт интерфейса жесткого диска.

EISA, Extended Industry Standard Architecture (расширенная стандартная промышленная архитектура) — тип шины.

EM64T, Extended Memory 64-bit Technology (расширенная технология работы с 64-битовой памятью) — группа команд в системе команд процессора Intel.

ENIAC, Electronic Numerical Integrator And Computer (электронно-цифровой интегратор и вычислитель) — название первой электронной вычислительной машины.

EPIC, Explicitly Parallel Instruction Computing (обработка команд с явным параллелизмом) — тип архитектуры системы команд.

EPROM, Erasable Programmable ROM (перепрограммируемые ПЗУ) — тип памяти компьютера.

ESCD, Extended Static Configuration Data (расширенные статические данные конфигурации) — тип памяти компьютера.

EUNet, European Unix Users Network — Европейская пользовательская Unix-сеть.

FDD, floppy disk drive — привод флоппи-дисков (гибких дисков).

FDDI, Fiber Distributed Data Interface — оптоволоконный интерфейс распределенных данных.

FEPROM, Flash EPROM (флэш-EPROM, программируемые ПЗУ) — тип памяти компьютера.

FireWire, fire wire (огненный провод) — тип шины.

Flops, Floating point operation per second (операций с плавающей точкой в секунду) — единица измерения производительности.

FMD-ROM, Fluorescent Multilayer Disk (многослойный флуоресцентный диск однократной записи) — перспективный тип оптического диска.

FPM DRAM, Fast Page Mode DRAM (динамическая память с быстрым страничным режимом) — тип микросхемы памяти.

FPU, Float Point Unit (блок плавающей точки) — сопроцессор.

FPU, Float Point Unit (устройство с плавающей точкой) — блок процессора.

FSB, Front Side Bus — системная шина, связывающая процессор с памятью или чипом.

FTP, File Transfer Protocol — протокол передачи файлов.

GDT, Global Descriptor Table — глобальная дескрипторная таблица.

GDTR, Global Descriptor Table Register — регистр глобальной дескрипторной таблицы.

HDD, Hard Disk Drive — привод жесткого диска.

HP, Hewlett-Packard — название фирмы, образовано от фамилий основателей компании.

HP, Hyper Treading (увеличенный поток, или сверхпоток) — архитектурная технология микропроцессора.

HPC, hand-held personal computer (на поверхности ладони, наладонный) — персональный компьютер.

HPM DRAM, Hyper Page Mode DRAM (динамическая память с гиперстраничным режимом) — тип микросхемы памяти.

http, Hyper Text Transfer Protocol (протокол передачи гипертекстов) — основной протокол Всемирной паутины.

I/O, input/output — ввод/вывод.

IA32, Intel Architecture 32 — 32-битовая архитектура микропроцессоров Intel.

IA64, Intel Architecture 64 — 64-битовая архитектура микропроцессоров Intel.

iAPX, Intel Advanced Processor Architecture (продвинутая процессорная архитектура) — часть названия модели компьютера.

IAS, Immediate Address Storage (память с прямой адресацией) — название машины, разработанной фон Нейманом.

IBM PC, International Business Machines Personal Computer — персональный компьютер фирмы IBM.

IBM PC/AT, International Business Machines Personal Computer Advanced Technology (IBM PC продвинутая технология) — тип персональных компьютеров.

IBM, International Business Machines corporation (корпорация Международные коммерческие машины) — название фирмы-производителя компьютеров.

IDE, Integrated Drive Electronics (интегрированное электронное устройство) — стандарт интерфейса жесткого диска.

IDE, Integrated Drive Electronics (интегрированные электронные устройства) — тип шины.

IEEE, Institute of Electrical and Electronics Engineers — Международный институт инженеров по электротехнике и электронике.

INTA, Interrupt Answer (ответ прерывания) — выход микропроцессора.

Intel, Integrated Electronics (объединенная электроника) — название фирмы-производителя микропроцессоров.

INTR, Interrupt Request (запрос на прерывание) — вход микропроцессора.

IP, Internet protocol — протокол сетевого уровня.

IPF, Itanium Processor Family — семейство процессоров Itanium.

IPX, Internet work Packet exchange — протокол сетевого уровня.

IPX/SPX, Sequenced Packed exchange/Internet work Packet exchange — стек коммуникационных протоколов фирмы Novell.

ISA, Industry Standard Architecture (стандартная промышленная архитектура) — тип шины.

ISDN, Integrated Services Digital Network — цифровая сеть с интегрированными службами.

ISO, International Organization for Standardization — международная организация по стандартам.

LAN, Local Area Network — локальная сеть.

LCD, Liquid-Crystal Display — жидкокристаллический дисплей.

LDT, Local Descriptor Table — локальная дескрипторная таблица.

LDTR, Local Descriptor Table Register — регистр локальной дескрипторной таблицы.

LFK, Livermore Fortran Kernels (ливерморские фортрановские ядра) — тест оценки производительности.

LQ, Letter Quality (машинописное качество) — уровень качества печати.

LRU, Least Recently Used (наиболее давно использованный) — алгоритм работы кэша.

MAN, Metropolitan Area Network — городская сеть.

MEMR#, Memory Read (чтение из памяти) — управляющий сигнал шины.

MESI, Modified Exclusive Shared Invalid (модифицированный, исключительный, разделенный, недействительный) — протокол работы кэша.

MIMD, Multiply Instruction Multiply Data stream (множественный поток команд, множественный поток данных) — класс вычислительных систем.

MIPS, Million Instructions Per Second (миллион машинных команд в секунду) — единица измерения производительности компьютера; семейство компьютеров.

MISD, Multiply Instruction Single Data stream (множественный поток команд, одиночный поток данных) — класс вычислительных систем.

mod, modification — параметр-модификатор, определяющий местоположение операнда.

MPC, Multimedia Personal Computer — мультимедийный персональный компьютер.

MPP, Massively Parallel Processor (процессор с массовым параллелизмом) — класс вычислительных систем.

MS DOS, Microsoft Disk Operation System — дисковая операционная система фирмы Microsoft.

NaN, Not a Number (нечисло) — специальный код вещественных чисел.

NC-NUMA, No Caching NUMA (системы NUMA без кэширования) — класс вычислительных систем.

NLQ, Near Letter Quality (близко к типографскому) — уровень качества печати.

NMI, Not Maskable Interrupt (немаскируемое прерывание) — вход микропроцессора.

NOW, Network Of WorkStation (сеть рабочих станций) — класс вычислительных систем.

NUMA, Non Uniform Memory Access (системы с неоднородным доступом к памяти) — класс вычислительных систем.

OLED, Organic Light Emitting Diodes (органические светодиоды) — тип дисплея.

OSI, Open System Interconnection (модель взаимодействия открытых систем) — эталонная модель построения компьютерных сетей.

PA Wide Word, Parallel Architecture Wide Word (всемирная параллельная архитектура) — тип архитектуры.

PA-RISC, Precision Architecture RISC (высокоточная архитектура RISC) — тип архитектуры.

PCI, Peripheral Component Interconnect (соединитель периферийных компонентов) — тип шины.

PDU, Protocol Data Unit (протокольный блок данных) — единица передачи по сети.

Pentium MMX, Multimedia Technology (мультимедийные технологии) — тип микропроцессоров.

PnP, Plug and Play (подключи и работай) — тип устройств компьютера.

POP, Post Office Protocol (протокол почтового отделения) — один из протоколов служб электронной почты.

PPP, Protocol Point to Point (протокол от точки к точке) — один из сетевых протоколов.

PROM, Programmable ROM (однократно программируемые ПЗУ) — тип памяти компьютера.

PSP, program segment prefix — префикс программного сегмента.

QAM-64, Quadrature Amplitude Modulation — метод квадратурной амплитудной модуляции.

r/m, register/memory — параметр регистр/память, уточняющий адресацию операнда.

R/W, Readable/Writeable (читаемый/записываемый) — часть дескриптора сегмента памяти.

RAID, Redundant Array of Inexpensive Disks (избыточный массив недорогих дисков, или Redundant Array of Independent Disks — избыточный массив независимых дисков) — стандарт интерфейса жесткого диска.

RAM, Random Access Memory — память произвольного доступа.

RAS#, Row Address Strobe (строб адреса строки) — управляющий сигнал микросхемы памяти.

RAW, Read After Write (чтение после записи) — вид взаимозависимости между машинными командами.

RDRAM, Rambus DRAM (от названия фирмы-производителя Rambus, Inc.) — тип микросхемы памяти.

Relarn, Russian Electronic Academic Research Network — российская академическая исследовательская электронная сеть.

Relcom, Reliable Communications (надежные коммуникации) — территориальная сеть в России.

RGB, Red, Green, Blue (красный, зеленый, синий) — метод цветной растровой графики.

RISC, Reduced Instruction Set Computer или Code (компьютер с сокращенным набором инструкций или усеченный набор кодов команд) — тип архитектуры системы команд.

ROM, Read Only Memory — память только для чтения.

RW, Read/Write (чтение/запись) — управляющий сигнал микросхемы памяти.

SAS, Serial Attached SCSI (последовательное соединение для интерфейса SCSI) — стандарт интерфейса жесткого диска.

SCSI, Small Computer System Interface (интерфейс малых вычислительных систем) — стандарт интерфейса жесткого диска.

SCSI, Small Computer System Interface (интерфейс малых вычислительных систем) — тип шины.

SDRAM, Synchronous DRAM (синхронизированная динамическая память) — тип микросхемы памяти.

SGI, Silicon Graphics International — название фирмы-производителя вычислительных систем.

SIMD, Single Instruction Multiple Data (одна инструкция — много данных) — класс вычислительных систем.

SIMD, Single Instruction Multiply Data stream (одиночный поток команд, множественный поток данных) — класс вычислительных систем.

SIMM, Single In Line Memory Modules — однорядные модули памяти.

SISD, Single Instruction Single Data stream (одиночный поток команд, одиночный поток данных) — класс вычислительных систем.

SLED, Single Large Expensive Disk (одиночный большой дорогой диск) — тип жесткого диска.

SLQ, Super LQ (суперкачественный) — уровень качества печати.

SMP, Symmetric Multi-Processing (симметричная многопроцессорность) — класс вычислительных систем.

SMTP, Simple Mail Transfer Protocol (простой протокол передачи почты) — один из протоколов служб электронной почты.

SP PA, Super Parallel Processor Architecture (суперпараллельная процессорная архитектура) — тип архитектуры.

SPARC, Scalable Processor ARChitecture (масштабируемая или наращиваемая архитектура процессора) — тип системы команд.

SPEC, Standard Performance Evaluation Corporation — корпорация, разрабатывающая стандартные тесты оценки производительности вычислительных систем.

SPX, Sequenced Packed exchange — протокол транспортного уровня.

SRAM, Static Random Access Memory — статическая память с произвольным доступом.

SSE, Streaming SIMD Extensions (потоковое расширение SIMD) — группа команд в системе команд процессора Intel.

STP, shielded twisted pair (экранированная витая пара) — тип кабеля.

SUN SPARC — семейство процессоров.

SUN, Stanford University Network — компьютерная сеть Стэнфордского университета.

SVGA, Super Video Graphics Array (супервидеографический массив) — тип адаптера.

TCP, Transmitting control protocol — протокол транспортного уровня.

TCP/IP, Transmitting control protocol/Internet protocol — стек коммуникационных проколов сети Интернет.

Telnet, Terminal over Network protocol — протокол службы удаленного сетевого доступа.

UMA, Uniform Memory Access (унифицированный однородный доступ к памяти) — класс вычислительных систем.

UPA, Ultra Port Architecture (архитектура сверхпорта, высокоскоростной коммутатор) — тип архитектуры.

USB, Universal Serial Bus (универсальная последовательная шина) — тип шины.

UTP, unshielded twisted pair (неэкранированная витая пара) — тип кабеля.

VGA, Video Graphics Array (видеографический массив) — тип адаптера.

VLIW, Very Large Instruction Word (очень длинное командное слово) — тип архитектуры системы команд.

VTOC, Volume Table Of Content (таблица содержания) — оглавление оптического диска.

W — признак длины операнда.

WAN, Wide Area Network — территориальная сеть.

WAR, Write After Read (запись после чтения) — вид взаимозависимости между машинными командами.

WAV, WAVeform-audio (волновая форма аудио) — формат кодирования звука.

WEW, Write After Write (запись после записи) — вид взаимозависимости между машинными командами.

Wi-Fi, Wireless Fidelity (беспроводная безукоризненная точность) — беспроводной метод доступа.

WiMax, Wireless Maximum — беспроводной метод доступа на больших расстояниях.

WORM, Write Once/Read Many (однократная запись, множественное считывание) — тип оптического диска.

АВМ — аналоговые вычислительные машины.

АЛУ — арифметико-логическое устройство.

БИС — большая интегральная схема.

БЭСМ — большая электронно-счетная машина.

ВЗУ — внешние запоминающие устройства.

ВП — внешняя память.

ВУ — внешнее устройство.

ГВМ — гибридные вычислительные машины.

ГМД — гибкий магнитный диск.

ДНФ — дизъюнктивная нормальная форма.

ЖК, жидкокристаллический — тип дисплея.

ЖМД — жесткий магнитный диск.

ИС — интегральная схема.

КМОП, комплементарный металл–оксид–полупроводник — тип памяти компьютера.

КОП — код операции.

КОПД — дополнительный код операции, уточняющая часть кода.

КПК — карманный персональный компьютер.

МИС — малая интегральная схема.

МКМД, множественный поток команд, множественный поток данных — класс вычислительных систем.

МКОД, множественный поток команд, одиночный поток данных — класс вычислительных систем.

МЭСМ — малая электронно-счетная машина.

НГМД — накопитель на гибких магнитных дисках.

ОЗУ — оперативное запоминающее устройство.

ОКМД, одиночный поток команд, множественный поток данных — класс вычислительных систем.

ОКОД, одиночный поток команд, одиночный поток данных — класс вычислительных систем.

ООП — основная оперативная память.

ОП — основная память.

ОП — оперативная память.

ПЗУ — постоянное запоминающее устройство.

ПМ — процессорный модуль.

ПУ — процессорный узел.

ПЭ — процессорный элемент.

СБИС — сверхбольшая интегральная схема.

СИС — средняя интегральная схема.

УВВ — устройства ввода/вывода.

УУ — устройство управления.

ЦВМ — дискретные или цифровые вычислительные машины.

ЦП — центральный процессор.

ЭВМ — электронная вычислительная машина.

ЭЛТ, электронно-лучевая трубка — тип дисплея.

Литература

1. *Абель П.* Язык Ассемблера для IBM PC и программирования. — М.: Высш. шк., 1992.
2. *Аладьев В. З., Хунт Ю. Я., Шишаков М. Л.* Основы информатики. — М.: Филинъ, 1999.
3. *Андреева Е., Фалина И.* Системы счисления и компьютерная арифметика. — М.: Лаборатория базовых знаний, 2000.
4. *Андерсон Д.* Дискретная математика и комбинаторика. — М.: Вильямс, 2003.
5. *Апокин И. А., Майстеров Л. Е.* История вычислительной техники. — М.: Наука, 1990.
6. Архитектура компьютерных систем и сетей / *Барановская Т. П., Лойко В. И., Семенов М. И., Трубилин А. И.*. — М.: Финансы и статистика, 2003.
7. *Блейхуд Р.* Теория и практика кодов, контролирующих ошибки. — М.: Мир, 1988.
8. Архитектура и топологии многопроцессорных вычислительных систем / *Богданов А. В., Корхов В. В., Мареев В. В., Станкова Е. Н.* — М.: Интернет-университет информационных технологий, 2004.
9. *Брой М.* Информатика. Основополагающее введение. — М.: Диалог-МИФИ, 1996.
10. *Вернер М.* Основы кодирования. — М.: Техносфера, 2004. (Сер. «Мир программирования»)
11. *Воеводин В. В., Воеводин Вл. В.* Параллельные вычисления. — СПб.: БХВ-Петербург, 2002.
12. *Головкин Б. А.* Параллельные вычислительные системы. — М.: Наука, 1980.
13. *Григорьев В. Л.* Микропроцессор i486. Архитектура и программирование: В 4 кн. — М.: Грандал, 1993.
14. *Гук М.* Аппаратные средства IBM PC. — СПб.: Питер, 2000.

15. *Гук М.* Аппаратные средства локальных сетей. — СПб.: Питер, 2002.
16. *Иртегов Д. В.* Введение в сетевые технологии. — СПб.: БХВ-Петербург, 2004.
17. *Ключко В. И.* Кодирование информации: Курс лекций / КубГТУ. — Краснодар, 1998.
18. *Корнеев В. В.* Современные микропроцессоры. — М.: Нолидж, 2003.
19. *Корнеев В. В.* Параллельные вычислительные системы. — М.: Нолидж, 1998.
20. *Королев Л. Н.* Структуры ЭВМ и их математическое обеспечение. — М.: Наука, 1978.
21. *Королев Л. Н., Миков А. И.* Информатика. Введение в компьютерные науки. — М.: Высш. шк., 2003.
22. *Лебедев А. Н.* Курс аналоговых вычислительных машин / ЛЭТИ. — Л., 1970.
23. *Майерс Г.* Архитектура современных ЭВМ (в 2 т.). — М.: Мир, 1985.
24. *Морс С. П., Алберт Д. Д.* Архитектура микропроцессора 80286. — М.: Радио и связь, 1990.
25. *Олифер В. Г., Олифер Н. А.* Компьютерные сети. — СПб.: Питер, 2001.
26. Рекомендации по преподаванию информатики в университетах. Computing Curricula 2001: Computer Science. — СПб.: Изд-во СПбГУ, 2002.
27. *Смирнов А. Д.* Архитектура вычислительных систем. — М.: Наука, 1990.
28. *Стариченко Б. Е.* Теоретические основы информатики. — М.: Горячая линия — Телеком, 2003.
29. *Таненбаум Э.* Архитектура компьютера. — 4-е изд. — СПб.: Питер, 2002.
30. *Таненбаум Э.* Компьютерные сети. — 4-е изд. — СПб.: Питер, 2005.
31. *Таненбаум Э., Ван Стен М.* Распределенные системы. Принципы и парадигмы. — СПб.: Питер, 2003.
32. *Юров В.* Assembler. — 2-е изд. — СПб.: Питер, 2003.
33. *Юров В.* Assembler: Практикум. — СПб.: Питер, 2001.
34. *Угрюмов Е. П.* Цифровая схемотехника. — СПб.: БХВ-Петербург, 2004.
35. *Хамахер К., Вранешич З., Заки С.* Организация ЭВМ. — СПб.: Питер, 2003.

Периодические издания

1. Ежемесячный журнал «Компьютер пресс», ISSN 0868-6157 (www.compress.ru).
2. Ежемесячный журнал «Мир ПК», ISSN 0235-3520 (www.pcworld.ru).
3. Ежемесячный журнал «Открытые системы. СУБД», ISSN 1028-7493 (www.osmag.ru).
4. Международный компьютерный еженедельник «Computerworld Россия», ISSN 1560-5213 (www.computerworld.ru).

Интернет-ресурсы

1. Ferra. Аналитические обзоры компьютеров и комплектующих, новости и цены компьютерного рынка // www.ferra.ru.
2. HardwarePortal. Обзоры и тесты компьютерного железа // www.hardwareportal.ru.
3. The collection of computer Science Bibliographies // liinwww.ira.uka.de/bibliography/index.html.
4. Whatis.ru. Все о компьютерах // www.whatis.ru.
5. Виртуальный компьютерный музей // www.computer_museum.ru.
6. *Воеводин В. В.* Параллельная обработка данных. Курс лекций. — 2001 // www.citforum.ru; www.parallel.ru.
7. Высокопроизводительные компьютеры // parallel.ru/computers.
8. Домашняя страница AMD // www.amd.com/ru-ru.
9. Железная столица. Новости аппаратных средств // www.stolica.ru.
10. Интернет-магазин компьютеров и оргтехники // www.xform.ru/default.asp.
11. Компьютерный музей // www.microsoft.com/museum.
12. *Левин В. К.* Отечественные суперкомпьютеры семейства МВС. Лаборатория параллельных информационных технологий НИВЦ МГУ // parallel.ru/mvs/levin.html.
13. Мир HARDware. Справочно-информационный сайт о компьютерной технике. Новости, обзоры, статьи об аппаратном и программном обеспечении компьютера. Тематические ссылки // www.hardw.com.ua.
14. Сайт компьютерных новостей. Техническая информация по аппаратному обеспечению: компьютерные новости, обзоры, советы и рекомендации и др. Ссылки на страницы производителей и поставщиков // www.ixbt.com.
15. Список 500 мощнейших суперкомпьютеров // www.top500.org/lists/plists.php?TB=1&M=11&Y=2004.
16. Тематический портал о компьютерах // hardvision.ru.
17. *Шнитман В.* Современные высокопроизводительные компьютеры. Центр информационных технологий. — 1998 // www.citmgu.ru; Электронное пособие // support.vologda.ru/Book/ARCHITECTURE/Svk/contents.htm.

Алфавитный указатель

В

Г

Д

Е

Ж

З

И

К

Л

М

Н

О

П

P

C

Т

У

Ф

Анатолий Николаевич Степанов

Архитектура вычислительных систем и компьютерных сетей

Заведующий редакцией	*А. Кривцов*
Руководитель проекта	*В. Шачин*
Литературные редакторы	*И. Кноп, Н. Рощина*
Иллюстрации	*Е. Дьяченко*
Корректоры	*А. Моносов, И. Смирнова*
Верстка	*Р. Гришанов*

Подписано в печать 20.10.06. Формат 70×100/16. Усл. п. л. 41,28. Тираж 3000. Заказ 3002.
ООО «Питер Пресс», 198206, Санкт-Петербург, Петергофское шоссе, 73, лит. А29.
Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.
Отпечатано по технологии CtP в ОАО «Печатный двор» им. А. М. Горького.
197110, Санкт-Петербург, Чкаловский пр., 15.