УЧЕБНИК ДЛЯ ВУЗОВ

Б. Я. Цилькер, С. А. Орлов

ОРГАНИЗАЦИЯ ЭВМ И СИСТЕМ

Допущено Министерством образования Российской Федерации в качестве учебника для студентов высших учебных заведений, обучающихся по направлению «Информатика и вычислительная техника»

300.piter.com

Издательская программа

300 лучших учебников для высшей школы в честь 300-летия Санкт-Петербурга

осуществляется при поддержке Министерства образования РФ

ПИТЕР

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

2004

Краткое содержание

Содержание

Введение

Мы живем в информационную эпоху: документы ЮНЕСКО свидетельствуют, что сейчас в информационной сфере занято больше половины населения развитых стран. Основу современных информационных технологий, их базис, составляют аппаратные средства компьютерной техники.

Современные вычислительные машины (ВМ) и системы (ВС) являются одним из самых значительных достижений научной и инженерной мысли, влияние которого на прогресс во всех областях человеческой деятельности трудно переоценить. Поэтому понятно то пристальное внимание, которое уделяется изучению ВМ и ВС в направлении «Информатика и вычислительная техника» высшего профессионального образования.

В государственном образовательном стандарте высшего профессионального образования содержание дисциплины «Организация ЭВМ и систем» определено следующим образом:

- основные характеристики, области применения ЭВМ различных классов;
- функциональная и структурная организация процессора;
- организация памяти ЭВМ;
- основные стадии выполнения команды;
- организация прерываний в ЭВМ;
- организация ввода-вывода;
- периферийные устройства;
- архитектурные особенности организации ЭВМ различных классов;
- параллельные системы;
- понятие о многомашинных и многопроцессорных вычислительных системах.

Все эти вопросы Освещены в учебнике, который вы держите в руках, уважаемый читатель. Иными словами, данный учебник отвечает всем требованиям образовательного стандарта.

Авторы стремились к достижению трех целей:

- изложить классические основы, демонстрирующие накопленный отечественный и мировой опыт вычислительных машин и систем;
- показать последние научные и практические достижения, характеризующие динамику развития аппаратных средств компьютерной техники;

- обобщить и отразить 25-летний университетский опыт преподавания авторами учебника соответствующих дисциплин.

Первая глава учебника посвящена базовым положениям. Обсуждаются понятия «организация» и «архитектура» вычислительных машин и систем, уровни абстракции, на которых эти понятия могут быть раскрыты. Прослеживается эволюция ВМ и ВС как последовательности идей, предопределивших современное состояние в области вычислительной техники. Анализируются тенденции дальнейшего развития архитектуры ВМ и ВС с учетом технологического прогресса и последних достижений в проектировании вычислительных средств.

Во второй главе дается понятие архитектуры системы команд и обсуждаются различные аспекты этой архитектуры. Рассматриваются основные виды информации, являющейся объектом обработки и хранения в ВМ и ВС. Приводятся основные способы представления такой информации: форматы, стандарты, размещение в памяти, способы доступа к данным. Представлены классификация и характеристика команд ВМ. Обсуждаются принципы выбора эффективной системы операций и системы адресации.

Третья глава является основой для понимания принципов функционирования вычислительных машин с классической фон-неймановской архитектурой. На примере гипотетической ВМ прослеживается взаимодействие узлов вычислительной машины в ходе выполнения типовых команд. Приводится описание языка микропрограммирования как средства формальной записи вычислительных процессов на уровне архитектуры ВМ.

Четвертая глава отведена принципам организации системы коммуникаций между элементами структуры ВМ. Дается понятие системной шины. Рассматриваются способы синхронизации и арбитража устройств, подключенных к шине.

В пятой главе определены принципы и средства, используемые при построении систем памяти ВМ. Поясняется концепция иерархического построения памяти. В первой части главы обсуждаются вопросы организации внутренней памяти с учетом ее реализации на базе полупроводниковых запоминающих устройств (ЗУ): структура памяти с произвольным доступом, матричная организация микросхем ЗУ, основные типы оперативных и постоянных запоминающих устройств. Описываются архитектурные аспекты внутренней памяти ВМ — модульное построение, конвейеризация, расслоение, обнаружение и исправление ошибок. Значительное внимание уделено принципам организации и функционирования кэш-памяти. Обсуждаются вопросы виртуализации памяти ВМ, методы и средства защиты памяти от несанкционированного доступа. Вторая часть главы содержит краткую характеристику различных типов внешних запоминающих устройств, включая магнитные и оптические дисковые ЗУ, магнитоленточные запоминающие устройства. Приводится классификация и описание массивов магнитных дисков с избыточностью (RAID).

Содержание шестой главы — это описание принципов организации устройств управления (УУ) ВМ. Обсуждаются вопросы построения, функционирования и проектирования УУ с «жесткой» логикой и УУ с микропрограммной организацией, а также способы ускорения их работы.

Предметом внимания седьмой главы являются операционные устройства ВМ. Рассматриваются жесткие и магистральные структуры операционных устройств, их организация и классификация, способы реализации в ВМ основных арифметических и логических операций с учетом обработки данных в различных формах представления и форматах. Наряду со «стандартными» способами реализации арифметических операций обсуждаются и такие алгоритмы, использование которых ведет к существенному ускорению вычислений.

Восьмая глава учебника посвящена вопросам организации систем ввода/вывода (СВВ). Рассматриваются способы организации ввода/вывода (программно-управляемый ввод/вывод, ввод/вывод по прерываниям, прямой доступ к памяти) и их влияние на эволюцию принципов построения СВВ. Описываются особенности систем ввода/вывода больших универсальных ВМ с их концепцией процессоров (каналов) ввода/вывода.

В девятой главе излагаются вопросы, касающиеся архитектуры процессоров вычислительных машин. Дается понятие конвейера команд, обсуждаются принципы организации такого конвейера и проблемы, возникающие при его реализации. Особое внимание уделяется конфликтам в конвейере команд и способам борьбы с ними. Поясняется концепция суперконвейеризации. Рассматривается проблема семантического разрыва и способы его преодоления в ВМ с архитектурами CISC и RISC. Глава завершается изложением концепции суперскалярного процессора.

Десятая глава предваряет вторую часть учебника, посвященную вопросам построения вычислительных систем, реализующих концепцию распараллеливания вычислений. Излагается теоретический базис таких вычислений. Приводится схема классификации параллельных вычислительных систем.

В одиннадцатой главе рассматриваются два основных принципа организации памяти ВС: общая (совместно используемая) память и распределенная память. Рассказывается об особенностях различных моделей как той, так и другой памяти. Значительное внимание в главе уделено вопросам когерентности кэш-памяти.

Двенадцатая глава содержит достаточно подробный обзор топологий сетей межсоединений, связывающих между собой компоненты вычислительных систем.

В тринадцатой главе сосредоточен материал по системам, которые согласно классификации Флинна можно отнести к системам класса SIMD. Несмотря на достаточную расплывчатость границ того или иного класса, допускаемую классификацией в схеме Флинна, в учебнике к SIMD-системам отнесены: векторные и векторно-конвейерные ВС, матричные ВС, ассоциативные ВС, вычислительные системы с систолической структурой и ВС с командным словом сверхбольшой длины.

В четырнадцатой главе рассматриваются системы класса MIMD. К таким в учебнике отнесены симметричные мультипроцессорные системы (SMP), кластерные ВС, системы с массовым параллелизмом (MPP), ВС на базе транспьютеров, системы с неоднородным доступом к памяти, ВС с обработкой по принципу волнового фронта.

В пятнадцатой главе описываются системы с нетрадиционным способом управления вычислительным процессом: потоковые и макропотоковые ВС, а также вычислительные системы с управлением по запросу. Именно особенность управ-

ления вычислениями предопределила выделение этих систем в отдельную группу, хотя в принципе их можно трактовать как варианты систем класса MIMD. В главе рассматривается также концепция гиперпотоковых вычислений; впрочем, изложение данного вопроса было бы уместно и в девятой главе.

Учебник предназначен для студентов инженерного, бакалаврского и магистерского уровней компьютерных специальностей, может быть полезен аспирантам, преподавателям и разработчикам вычислительных машин и систем.

Вот, пожалуй, и все. Насколько удалась эта работа - судить гам,--ув'аэнаемый читатель.

Благодарности

Прежде всего, наши Прежде всего, наши слова искренней любви родителям.

Самые теплые слова благодарности нашим семьям, родным, любимым и близким людям. Без их долготерпения, внимания, поддержки, доброжелательности и сердечной заботы эта книга никогда бы не была написана.

Выход в свет данной работы был бы невозможен вне творческой атмосферы, бесчисленных вопросов и положительной обратной связи, которую создавали наши многочисленные студенты.

Хочется отметить особо значимую роль руководителя проекта Юрия Суркиса, благодаря незаурядным профессиональным качествам которого и состоялось данное издание. Авторы искренне признательны всем талантливым сотрудникам издательства «Питер».

И, конечно, огромная признательность нашим коллегам, общение с которыми поддерживало огонь творчества, и нашим учителям, давшим базис образования, подкрепляемого нами всю жизнь.

да От издательства

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

• М будем рады узнать ваше мнение!

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

Глава 1

Становление и эволюция цифровой вычислительной техники

Изучение любого вопроса принято начинать с **договоренностей** о терминологии. В нашем случае определению подлежат понятия *вычислительная машина* (ВМ) и *вычислительная система* (ВС). Сразу же оговорим, что предметом рассмотрения будут исключительно цифровые машины и системы, то есть устройства, оперирующие дискретными величинами. В литературе можно найти множество самых различных определений терминов «вычислительная **машина**» и «вычислительная система».. Причина такой терминологической неопределенности кроется в невозможности дать удовлетворяющее всех четкое определение, достойное роли стандарта. Любая из известных формулировок несет в себе стремление авторов отразить наиболее существенные, по их мнению, моменты, в силу чего не может быть всеобъемлющей. В подтверждение этого тезиса приведем несколько определений термина «**вычислительная** машина», взятых из различных литературных источников[1]. Итак, вычислительная машина - это:

1. Устройство, которое принимает данные, обрабатывает их в соответствии с хранимой программой, генерирует результаты и обычно состоит из блоков ввода, вывода, памяти, арифметики, логики и управления.
2. Функциональный блок, способный выполнять реальные вычисления, включающие множественные арифметические и логические операции, без участия человека в процессе этих вычислений.
3. Устройство, способное:
   - хранить программу или программы обработки и по меньшей мере информацию, необходимую для выполнения программы;
   - быть свободно перепрограммируемым в соответствии с требованиями пользователя;

[1] Две первые цитаты взяты из различных толковых словарей, а третья - из постановления о таможенном тарифе.

- D выполнять арифметические вычисления, определяемые пользователем;
- O выполнять без вмешательства человека программу обработки, требующую изменения действий путем принятия логических решений в процессе обработки.

Не отдавая предпочтения ни одной из известных формулировок терминов «вычислительная машина» и «вычислительная система», тем не менее воспользуемся наиболее общим их определением [33], условившись, что по мере необходимости смысловое их наполнение может уточняться.

Термином *вычислительная машина* будем обозначать комплекс технических и программных средств, предназначенный для автоматизации подготовки и решения задач пользователей.

В свою очередь, *вычислительную систему* определим как совокупность взаимосвязанных и взаимодействующих процессоров или вычислительных машин, периферийного оборудования и программного обеспечения, предназначенную для подготовки и решения задач пользователей.

Таким образом, формально отличие ВС от ВМ выражается в количестве вычислителей. Множественность вычислителей позволяет реализовать в ВС параллельную обработку. С другой стороны, современные вычислительные машины с одним процессором также обладают определенными средствами распараллеливания вычислительного процесса. Иными словами, грань между ВМ и ВС часто бывает весьма расплывчатой, что дает основание там, где это целесообразно, рассматривать ВМ как одну из реализаций ВС. И напротив, вычислительные системы часто строятся из традиционных ВМ и процессоров, поэтому многие из положений, относящихся к ВМ, могут быть распространены и на ВС. Последнее замечание имеет непосредственное отношение к материалу данной главы. Условимся, что сказанное относительно вычислительных машин распространяется и на ВС с традиционными процессорами. В тех случаях, когда излагаемый материал справедлив лишь по отношению ВС, структура или принцип действия которых отличается от традиционных, будет действовать термин «вычислительная система».

И, наконец, заключительное замечание. Специфика главы вынуждает использовать в ней многие понятия, полное содержание которых станет ясным лишь после изучения последующих разделов книги. Там, где это возможно, пояснения будут даваться по ходу изложения (правда, в упрощенном виде). В любом случае, для получения полной картины к материалу данной главы имеет смысл еще раз вернуться после ознакомления со всей книгой.

Определение понятия «архитектура»

Рассмотрение принципов построения и функционирования вычислительных машин и систем предварим определением термина *архитектура* в том виде, как он будет трактоваться в данной книге.

Под архитектурой вычислительной машины обычно понимается логическое построение ВМ, то есть то, какой машина представляется программисту. Впервые термин «архитектура вычислительной машины» (computer architecture) был употреблен фирмой IBM при разработке машин семейства IBM 360 для описания тех

средств, которыми может пользоваться программист, составляя программу на уровне машинных команд. Подобную трактовку называют «узкой», и охватывает она перечень и формат команд, формы представления данных, механизмы ввода/вывода, способы адресации памяти и т. п. Из рассмотрения выпадают вопросы физического построения вычислительных средств: состав устройств, число регистров процессора, емкость памяти, наличие специального блока для обработки вещественных чисел, тактовая частота центрального процессора и т. д. Этот круг вопросов принято определять понятием *организация* или *структурная организация.*

Архитектура (в узком смысле) и организация — это две стороны описания ВМ и ВС. Поскольку для наших целей, помимо теоретической строгости, такое деление не дает каких-либо преимуществ, то в дальнейшем будем пользоваться термином **«архитектура»,** правда, в «широком» его толковании, объединяющем как архитектуру в узком смысле, так и организацию ВМ. Применительно к вычислительным системам термин «архитектура» дополнительно распространяется на вопросы распределения функций между составляющими ВС и взаимодействия этих составляющих.

Уровни детализации структуры вычислительной машины

Вычислительная машина как законченный объект являет собой плод усилий специалистов в самых различных областях человеческих знаний. Каждый специалист рассматривает вычислительную машину с позиций стоящей перед ним задачи, абстрагируясь от несущественных, по его мнению, деталей. В табл. 1.1 перечислены специалисты, принимающие участие в создании ВМ, и круг вопросов, входящих в их компетенцию.

Таблица 1.1. Распределение функций между разработчиками вычислительной машины

Специалист	Круг вопросов
Производитель полупроводниковых материалов	Материал для интегральных микросхем (легированный кремний, диоксид кремния и т. п.)
Разработчик электронных схем	Электронные схемы узлов ВМ (разработка и анализ)
Разработчик интегральных микросхем	Сверхбольшие интегральные микросхемы (схемы электронных элементов, их размещение на кристалле)
Системный архитектор	Архитектура и организация вычислительной машины (устройства и узлы, система команд и т. п.)
Системный программист	Операционная система, компиляторы
Теоретик	Алгоритмы, абстрактные структуры данных

, Круг вопросов, рассматриваемых в данном курсе, по большей части относится к компетенции системного архитектора и охватывает различные степени детализации ВМ и ВС. В принципе таких уровней может быть достаточно много, однако сложившаяся практика ограничивает их число четырьмя уровнями (рис. 1.1);

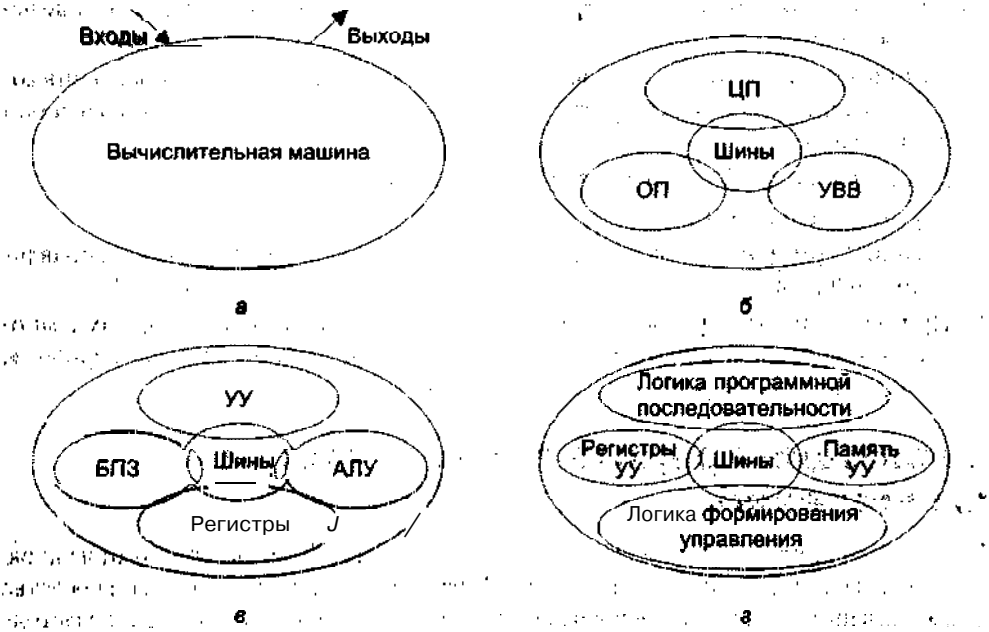


рис. 1.1. Уровни детализации вычислительной машины: *а* — уровень «черного ящика»; б — уровень общей архитектуры; в — уровень архитектуры центрального процессора; г - уровень архитектуры устройства управления

На первом уровне вычислительная машина рассматривается как устройство, способное хранить и обрабатывать информацию, а также обмениваться данными с внешним миром (см. рис..1.1, а). ВМ представляется «черным ящиком», который может быть подключен к коммуникационной сети и к которому, в свою очередь, могут подсоединяться периферийные устройства.

Уровень общей архитектуры (см. рис. 1.1, *б*) предполагает представление ВМ в виде четырех составляющих: центрального процессора (ЦП), основной памяти (ОП), устройства ввода/вывода (УВВ) и системы шин.

На третьем уровне детализируется каждое из устройств второго уровня. Для примера взят центральный процессор (см. рис. 1.1, *в*) В простейшем варианте в нем можно выделить:

• арифметико-логическое устройство (АЛУ); обеспечивающее обработку целых чисел;

• блок обработки чисел в формате с плавающей защей запятой (БПЗ);

регистры процессора, использующиеся для краткосрочного хранения команд данных и адресов;

краткосрочного хранения ком

- устройство управления (УУ), обеспечивающее совместное функционирование устройств ВМ;
- внутренние шины.

На четвертом уровне детализируются элементы третьего уровня. Так, на рис. **1.1,** *г* раскрыта структура устройства управления. УУ представлено в виде четырех составляющих:

- логики программной последовательности — электронных схем, обеспечивающих выполнение команд программы в последовательности, предписываемой программой;
- регистров и дешифраторов устройства управления;
- управляющей памяти;
- логики формирования управления, генерирующей все необходимые управляющие сигналы.

Применительно к параллельным и распределенным многопроцессорным и многомашинным вычислительным системам зачастую вводят понятие **«метауровня».** На данном этапе метауровень рассматриваться не будет.

Эволюция средств автоматизации вычислений

Попытки облегчить, ав идеале автоматизировать процесс вычислений имеют давнюю историю, насчитывающую более 5000 лет. С развитием науки и технологий средства автоматизации вычислений непрерывно совершенствовались. Современное состояние вычислительной техники (ВТ) являет собой результат многолетней эволюции.

В последнее время вопросы развития ВТ стали предметом особо пристального внимания ученых, свидетельством чего служит активно развивающаяся новая область знаний, получившая название «Теория эволюции **компьютеров»** (Computer evolution theory). Создатели теории обратили внимание на сходство закономерностей эволюции вычислительной техники и эволюции в биологии. В основу новой науки положены следующие постулаты:

- самозарождение **«живых»** вычислительных систем из **«неживых»** элементов (в биологии это явление известно как *абиогенез);*
- поступательное продвижение по древу эволюции — от *протопроцессорных* (однопроцессорных) вычислительных машин к *полипроцессорным* (многопроцессорным) вычислительным системам;
- прогресс в технологии вычислительных систем как следствие полезных мутаций и вариаций;
- отмирание устаревших технологий в результате естественного отбора;
- закон Мура[1] как подтверждение эволюции вычислительных систем.

[1] **Гордон Мур — один из основателей компании Intel, в 1965 году сделал знаменательное наблюдение, позже получившее название закона Мура. Он заметил, что плотность транзисторов на кремниевой подложке удваивается каждые 18-24 месяца, соответственно в два раза растет их производительность и в два раза падает их рыночная стоимость.**

По мнению специалистов в области теории эволюции компьютеров, изучение закономерностей развития вычислительных машин и систем может, как и в биологии, привести к ощутимым практическим результатам.

В традиционной трактовке эволюцию вычислительной техники представляют как последовательную смену поколений ВТ. Появление термина «поколение» относится к 1964 году, когда фирма IBM выпустила серию компьютеров IBM 360, назвав эту серию «компьютерами третьего поколения». Сам термин имеет разные определения, наиболее популярными из которых являются:

- «Поколения вычислительных машин — это сложившееся в последнее время разбиение вычислительных машин на классы, определяемые элементной базой и производительностью» [30].
- «Поколения компьютеров — нестрогая классификация вычислительных систем по степени развития аппаратных и, в последнее время, программных средств» [37].

При описании эволюции ВТ обычно используют один из двух подходов: хронологический или технологический. В первом случае - это хронология событий, существенно повлиявших на становление ВТ. Для наших целей больший интерес представляет технологический подход, когда развитие вычислительной техники рассматривается в терминах архитектурных решений и технологий. По словам главного конструктора фирмы DEC и одного из изобретателей мини-ЭВМ Белла: - «История компьютерной индустрии почти всегда двигалась технологией».

В качестве узловых моментов, определяющих появление нового поколения ВТ, обычно выбираются революционные идеи или технологические прорывы, кардинально изменяющие дальнейшее развитие средств автоматизации вычислений. Одной из таких идей принято считать концепцию вычислительной машины с хранимой в памяти программой, сформулированную Джоном фон Нейманом. Взяв ее за точку отсчета, историю развития ВТ можно представить в виде трех этапов:

- донеймановского периода;
- эры вычислительных машин и систем с фон-неймановской архитектурой;
- постнеймановской эпохи — эпохи параллельных и распределенных вычислений, где наряду с традиционным подходом все большую роль начинают играть отличные от фон-неймановских принципы организации вычислительного процесса.

Значительно большее распространение, однако, получила привязка поколений к смене технологий. Принято говорить о «механической» эре (нулевое поколение) и последовавших за ней пяти поколениях ВС [210]. Первые четыре поколения традиционно связывают с элементной базой вычислительных систем: электронные лампы, полупроводниковые приборы, интегральные схемы малой степени интеграции (ИМС), большие (БИС), сверхбольшие (СБИС) и ультрабольшие (УБИС) интегральные микросхемы. Пятое поколение в общепринятой интерпретации ассоциируют не столько с новой элементной базой, сколько с интеллектуальными возможностями ВС. Работы по созданию ВС пятого поколения велись в рамках четырех достаточно независимых программ, осуществлявшихся учеными США, Японии, стран Западной Европы и стран Совета экономической взаимопомощи.

Ввиду того, что ни одна из программ не привела к ожидаемым результатам, разговоры о ВС пятого поколения понемногу утихают. Трактовка пятого поколения явно выпадает из «технологического» принципа. С другой стороны, причисление всех ВС на базе сверхбольших интегральных схем (СБИС) к четвертому поколению не отражает принципиальных изменений в архитектуре ВС, произошедших за последние годы. Чтобы в какой-то мере проследить роль таких изменений, воспользуемся несколько отличной трактовкой, предлагаемой в [174]. В работе выделяется шесть поколений ВС. Попытаемся кратко охарактеризовать каждое из них, выделяя наиболее значимые события.

Нулевое поколение (1492-1945)

Для полноты картины упомянем два события, произошедшие до нашей эры: первые счеты - абак, изобретенные в древнем Вавилоне за 3000 лет до н. э., и их более «современный» вариант с косточками на проволоке, появившийся в Китае примерно за 500 лет также до н. э.

«Механическая» эра (нулевое поколение) в эволюции ВТ связана с механическими, а позже - электромеханическими вычислительными устройствами. Основным элементом механических устройств было зубчатое колесо. Начиная с XX века роль базового элемента переходит к электромеханическому реле. Не умаляя значения многих идей «механической» эры, необходимо отметить, что ни одно из созданных устройств нельзя с полным основанием назвать вычислительной машиной в современном ее понимании. Чтобы подчеркнуть это, вместо термина «вычислительная машина» будем использовать такие слова, как «вычислитель», «калькулятор» и т. п.

Хронология основных событий «механической» эры выглядит следующим образом.

1492 год. В одном из своих дневников Леонардо да Винчи приводит рисунок тринадцатиразрядного десятичного суммирующего устройства на основе зубчатых колес.

1623 год. Вильгельм Шиккард (Wilhelm Schickard, 1592-1635), профессор университета Тюбингена, разрабатывает устройство на основе зубчатых колес («считающие часы») для сложения и вычитания шестиразрядных десятичных чисел. Было ли устройство реализовано при жизни изобретателя, достоверно неизвестно, но в 1960 году оно было воссоздано и проявило себя вполне работоспособным.

1642 год. Блез Паскаль (Blaise Pascal, 1623-1663) представляет «Паскалин» - первое реально осуществленное и получившее известность механическое цифровое вычислительное устройство. Прототип устройства суммировал и вычитал пятиразрядные десятичные числа. Паскаль изготовил более десяти таких вычислителей, причем последние модели оперировали числами длиной в восемь цифр.

1673 год. Готфрид Вильгельм Лейбниц (Gottfried Wilhelm Leibniz, 1646-1716) создает «пошаговый вычислитель» - десятичное устройство для выполнения всех четырех арифметических операций над 12-разрядными десятичными числами. Результат умножения представлялся 16 цифрами. Помимо зубчатых колес в устройстве использовался новый элемент - ступенчатый валик.

1786 год. Немецкий военный инженер Иоганн Мюллер (Johann Mueller, 1746-1830) выдвигает идею «разностной машины» — специализированного калькуля-

тора для табулирования логарифмов, вычисляемых разностным методом. Калькулятор, построенный на ступенчатых валиках Лейбница, получился достаточно небольшим (13 см в высоту и 30 см в диаметре), но при этом мог выполнять все четыре арифметических действия над 14-разрядными числами.

1801 год. Жозеф Мария Жаккард (Joseph-Marie Jacquard, 1752-1834) строит ткацкий станок с программным управлением, программа работы которого задается с помощью комплекта перфокарт.

1832 год. Английский математик Чарльз Бэббидж (Charles Babbage, 1792-1871) создает сегмент разностной машины, оперирующий шестиразрядными числами и разностями второго порядка. Разностная машина Бэббиджа по идее аналогична калькулятору Мюллера.

1834 год. Пер Георг Шутц (Per George Scheutz, 1785-1873) из Стокгольма, используя краткое описание проекта Бэббиджа, создает из дерева небольшую разностную машину.

1836 год. Бэббидж разрабатывает проект «аналитической машины». Проект предусматривает три считывателя с перфокарт для ввода программ и данных, память (по Бэббиджу - «склад») на пятьдесят 40-разрядных чисел, два аккумулятора для хранения промежуточных результатов. В программировании машины предусмотрена концепция условного перехода. В проект заложен также и прообраз микропрограммирования - содержание инструкций предполагалось задавать путем позиционирования металлических штырей в цилиндре с отверстиями. По оценкам автора, суммирование должно было занимать 3 с, а умножение и деление - 2-4 мин.

1843 год. Георг Шутц совместно с сыном Эдвардом (Edvard Scheutz, 1821-1881) строят разностную машину с принтером для работы с разностями третьего порядка.

1871 год. Бэббидж создает прототип одного из устройств своей аналитической машины — «мельницу» (так он окрестил то, что сейчас принято называть центральным процессором), а также принтер.

1885 год. Дорр Фельт (Dorr E. Felt, 1862-1930) из Чикаго строит свой «комптометр» — первый калькулятор, где числа вводятся нажатием клавиш.

1890 год. Результаты переписи населения в США обрабатываются с помощью перфокарточного табулятора, созданного Германом Холлеритом (Herman Hollerith, 1860-1929) из Массачусетсского технологического института.

1892 год. Вильям Барроуз (William S. Burroughs, 1857-1898) предлагает устройство, схожее с калькулятором Фельта, но более надежное, и от этого события берет старт индустрия офисных калькуляторов.

1937 год. Джорж Стибитц (George Stibitz, 1904-1995) из Bell Telephone Laboratories демонстрирует первый однобитовый двоичный вычислитель на базе электромеханических реле.

1937 год. Алан Тьюринг (Alan M. Turing, 1912-1954) из Кембриджского университета публикует статью, в которой излагает концепцию теоретической упрощенной вычислительной машины, в дальнейшем получившей название машины Тьюринга.

1938год. Клод Шеннон (Claude E. Shannon, 1916-2001) публикует статью о реализации символической логики на базе реле.

1938 год. Немецкий инженер Конрад Цузе (Konrad Zuse, 1910-1995) строит механический программируемый вычислитель Z1 с памятью на 1000 бит. В последнее время Z1 все чаще называют первым в мире компьютером.

1939 год. Джордж Стибитц и Сэмюэль Вильямс (Samuel Williams, 1911-1977) представили Model I – калькулятор на базе релейной логики, управлятмый с помощью модифицированного телетайпа, что позволило подключаться к калькулятору по телефонной линии. Более поздние модификации допускали также деленную степень программирования.

1940 год. Следующая работа Цузе — электромеханическая машина Z2, основу которой составляла релейная логика, хотя память, как и в Z1, была механической.

1941 год. Цузе создает электромеханический программируемый вычислитель Z3. Вычислитель содержит 2600 электромеханических реле. Z3 — это первая попытка реализации принципа программного управления, хотя и не в полном объеме (в общепринятом понимании этот принцип еще не был сформулирован). В частности, не предусматривалась возможность условного перехода. Программа хранилась на перфоленте. Емкость памяти составляла 64 22-битовых слова. Операция умножения занимала 3-5 с.

1943 год. Группа ученых Гарвардского университета во главе с Говардом Айкеном (Howard Aiken, 1900–1973) разрабатывает вычислитель ASCC Mark I (Automatic Sequence-Controlled Calculator Mark I). - первый программно управляемьш вычислитель, получивший широкую известность. Длина устройства составила 18 м, а весило оно 5 т. Машина состояла из множества вычислителей, обрабатывающих свои части общей задачи под управлением единого устройства управления. Команды считывались с бумажной перфоленты и выполнялись в порядке считьван ия. Данные считывались с перфокарт. Вычислитель обрабатывал 23-разрядные числа, при этом сложение занимало 0,3 с, умножение - 4 с, а деление- 10 с.

1945 год. Цузе завершает Z4 - улучшенную версию вычислителя Z3. По архитектуре у Z4 очень много общих черт с современными ВМ: память и процессор представлены отдельными устройствами, процессор может обрабатывать числа с плавающей запятой и, в дополнение к четырем основным арифметическим операциям, способен извлекать квадратный корень. Программа хранится на перфоленте и считывается последовательно.

Не умаляя важности каждого из перечисленных фактов, в качестве важнейшего момента «механической» эпохи все-таки выделим аналитическую машину Чарль Бэббиджа и связанные с ней идеи.

Первое поколение (1937–1953)

Нароль первой в истории электронной вычислительной машины в разные периоды претендовало несколько разработок. Общим у них было использование схем на базе электронно-вакуумных ламп вместо электромеханических реле, Предполагалось, что электронные ключи будут значительно надежнее, поскольку в них отсутствуют движущиеся части, однако технология того времени была настолько несовершенной, что, по надежности электронные лампы оказались ненамного лучше, чем реле. Однако у электронных компонентов имелось одно важное преимуще-

ство: выполненные на них ключи могли переключаться примерно в тысячу раз быстрее своих электромеханических аналогов.

Первой электронной вычислительной машиной чаще всего называют специализированный калькулятор ABC (Atanasoff-Berry Computer). Разработан он был в период с 1939 по 1942 год профессором Джоном Атанасовым (John V. Atanasoff, 1903-1995) совместно с аспирантом Клиффордом Берри (Clifford Berry, 1918-1963) и предназначался для решения системы линейных уравнений (до 29 уравнений с 29 переменными). ABC обладал памятью на 50 слов длиной 50 бит, а запоминающими элементами служили конденсаторы с цепями регенерации. В качестве вторичной памяти использовались перфокарты, где отверстия не перфорировались, а прожигались. ABC стал считаться первой электронной ВМ, после того как судебным решением были аннулированы патенты создателей другого электронного калькулятора - ENIAC. Необходимо все же отметить, что ни ABC, ни ENIAC не являются вычислительным машинами в современном понимании этого термина и их правильней классифицировать как калькуляторы.

Вторым претендентом на первенство считается вычислитель Colossus, построенный в 1943 году в Англии в местечке Bletchley Park близ Кембриджа. Изобретателем машины был профессор Макс Ньюмен (Max Newman, 1987-1984), а изготовил его Томми Флауэрс (Tommy Flowers, 1905-1998). Colossus был создан для расшифровки кодов немецкой шифровальной машины «Лоренц Шлюссель-цузат-40». В состав команды разработчиков входил также Алан Тьюринг. Машина была выполнена в виде восьми стоек высотой 2,3 м, а общая длина ее составляла 5,5 м. В логических схемах машины и в системе оптического считывания информации использовалось 2400 электронных ламп, главным образом тиратронов. Информация считывалась с пяти вращающихся длинных бумажных колец со скоростью 5000 символов/с.

Наконец, третий кандидат на роль первой электронной ВМ - уже упоминавшийся программируемый электронный калькулятор общего назначения ENIAC (Electronic Numerical Integrator and Computer — электронный цифровой интегратор и вычислитель). Идея калькулятора, выдвинутая в 1942 году Джоном Мочли (John J. Mauchly, 1907-1980) из университета Пенсильвании, была реализована им совместно с Преспером Эккертом (J. Presper Eckert, 1919-1995) в 1946 году. С самого начала ENIAC активно использовался в программе разработки водородной бомбы. Машина эксплуатировалась до 1955 года и применялась для генерирования случайных чисел, предсказания погоды и проектирования аэродинамических труб. ENIAC весил 30 тонн, содержал 18 000 радиоламп, имел размеры 2,5 x 30 м и обеспечивал выполнение 5000 сложений и 360 умножений в секунду. Использовалась десятичная система счисления. Программа задавалась схемой коммутации триггеров на 40 наборных полях. Когда все лампы работали, инженерный персонал мог настроить ENIAC на новую задачу, вручную изменив подключение 6000 проводов. При пробной эксплуатации выяснилось, что надежность машины чрезвычайно низка — поиск неисправностей занимал от нескольких часов до нескольких суток. По своей структуре ENIAC напоминал механические вычислительные машины. 10 триггеров соединялись в кольцо, образуя десятичный счетчик, который исполнял роль счетного колеса механической машины. Десять таких колец плюс

два триггера для представления знака числа представляли запоминающий регистр. Всего в ENIAC было 20 таких регистров. Система переноса десятков в накопителях была аналогична предварительному переносу в машине Бэббиджа.

При всей важности каждой из трех рассмотренных разработок основное событие, произошедшее в этот период, связано с именем Джона фон Неймана. Американский математик Джон фон Нейман (John von Neumann, 1903-1957) принял участие в проекте ENIAC в качестве консультанта. Еще до завершения ENIAC Эккерт, Мочли и фон Нейман приступили к новому проекту - EDVAC, главной особенностью которого стала идея *хранимой в памяти программы*.

Технология программирования в рассматриваемый период была еще на зачаточном уровне. Первые программы составлялись в машинных кодах — числах, непосредственно записываемых в память ВМ. Лишь в 50-х годах началось использование языка ассемблера, позволявшего вместо числовой записи команд использовать символьную их нотацию, после чего специальной программой, также называемой ассемблером, эти символьные обозначения транслировались в соответствующие коды.

Несмотря на свою примитивность, машины первого поколения оказались весьма полезными для инженерных целей и в прикладных науках. Так, Атанасофф подсчитал, что решение системы из восьми уравнений с восемью переменными с помощью популярного тогда электромеханического калькулятора Маршана заняло бы восемь часов. В случае же 29 уравнений с 29 переменными, с которыми калькулятор ABC справлялся менее чем за час, устройство с калькулятором Маршана затратило бы 381 час. С первой задачей в рамках проекта водородной бомбы ENIAC справился за 20 с, в противовес 40 часам, которые понадобились бы при использовании механических калькуляторов.

В 1947 году под руководством С. А. Лебедева начаты работы по созданию малой электронной счетной машины (МЭСМ). Эта ВМ была запущена в эксплуатацию в 1951 году и стала первой электронной ВМ в СССЗ и континентальной Европе.

В 1952 году Эккерт и Мочли создали первую коммерчески успешную машину UNIVAC. Именно с помощью этой ВМ было предсказано, что Эйзенхауэр в результате президентских выборов победит Стивенсона с разрывом в 438 голосов (фактический разрыв составил 442 голоса).

Также в 1952 году в опытную эксплуатацию была запущена вычислительная машина М-1 (И. С. Брук, Н. Я. Матюхин, А. Б. Залкинд). М-1-содержала 730 электронных ламп, оперативную память емкостью 256 25-разрядных слов, рулонный телетайп и обладала производительностью 15-20 операций/с. Впервые была применена двухадресная система команд. Чуть позже группой выпускников МЭИ под руководством И. С. Брука создана машина М-2 с емкостью оперативной памяти 512 34-разрядных слов и быстродействием 2000 операций/с.

В апреле 1953 года в эксплуатацию поступила самая быстродействующая в Европе ВМ БЭСМ (С. А. Лебедев). Ее быстродействие составило 8000-10 000 операций/с. Примерно в то же время выпущена ламповая ВМ «Стрела» (Ю. А. Базилевский, Б. И. Рамееев) с быстродействием 2000 операций/с.

Второе поколение (1954-1962)

Второе поколение характеризуется рядом достижений в элементной базе, структуре и программном обеспечении. Принято считать, что поводом для выделения нового поколения ВМ стали технологические изменения, и, главным образом, переход от электронных ламп к полупроводниковым диодам и транзисторам со временем переключения порядка 0,3 мс.

Первой ВМ, выполненной полностью на полупроводниковых диодах и транзисторах, стала TRADIC (TRAnisitor Digital Computer), построенная в Bell Labs по заказу военно-воздушных сил США как прототип бортовой ВМ. Машина состояла из 700 транзисторов и 10 000 германиевых диодов. За два года эксплуатации TRADIC отказали только 17 полупроводниковых элементов, что говорит о прорыве в области надежности, по сравнению с машинами на электронных лампах. Другой достойной упоминания полностью полупроводниковой ВМ стала ТХ-0, созданная в 1957 году в Массачусетсском технологическом институте.

Со вторым поколением ВМ ассоциируют еще одно принципиальное технологическое усовершенствование — переход от устройств памяти на базе ртутных линий задержки к устройствам на магнитных сердечниках. В запоминающих устройствах (ЗУ) на линиях задержки данные хранились в виде акустической волны, непрерывно циркулирующей по кольцу из линий задержки, а доступ к элементу данных становился возможным лишь в момент прохождения соответствующего участка волны вблизи устройства считывания/записи. Главным преимуществом ЗУ на магнитных сердечниках стал произвольный доступ к данным, когда в любой момент доступен любой элемент данных, причем время доступа не зависит от того, какой это элемент.

Технологический прогресс дополняют важные изменения в архитектуре ВМ. Прежде всего, это касается появления в составе процессора ВМ индексных регистров, что позволило упростить доступ к элементам массивов. Прежде, при циклической обработке элементов массива, необходимо было модифицировать код команды, в частности хранящийся в нем адрес элемента массива. Как следствие, в ходе вычислений коды некоторых команд постоянно изменялись, что затрудняло отладку программы. С использованием индексных регистров адрес элемента массива вычисляется как сумма адресной части команды и содержимого индексного регистра. Это позволяет обратиться к любому элементу массива, не затрагивая код команды, а лишь модифицируя содержимое индексного регистра.

Вторым принципиальным изменением в структуре ВМ стало добавление аппаратного блока обработки чисел в формате с плавающей запятой. До этого обработка вещественных чисел производилась с помощью подпрограмм, каждая из которых имитировала выполнение какой-то одной операции с плавающей запятой (сложение, умножение и т. п.), используя для этой цели обычное целочисленное арифметико-логическое устройство.

Третье значимое нововведение в архитектуре ВМ — появление в составе вычислительной машины процессоров ввода/вывода, позволяющих освободить центральный процессор от рутинных операций по управлению вводом/выводом и обеспечивающих более высокую пропускную способность тракта «память - устройства ввода/вывода» (УВВ).

Ко второму поколению относятся и две первые суперЭВМ, разработанные для ускорения численных вычислений в научных приложениях. Термин «суперЭВМ» первоначально применялся по отношению к ВМ, производительность который на один или более порядков превосходила таковую для прочих вычислительнь машин того же поколения. Во втором поколении этому определению отвечали две ВМ (правильнее сказать системы): LARC (Livermore Atomic Research Computer)' и IBM 7030. Помимо прочего, в этих ВМ нашли воплощение еще две новинки со-- вмещение операций процессора с обращением к памяти и простейшие формы параллельной обработки данных.

Заметным событием данного периода стало появление в 1958году машины М-20. В этой ВМ, в частности, были реализованы: частичное совмещение операций, ап паратные средства поддержки программных циклов, возможность параллельной работы процессора и устройства вывода. Оперативная память емкостью 4096 45-разрядных слов была выполнена на магнитных сердечниках.

Шестидесятые оды XX века стали периодом бурного развития вычислительной техники в СССР. За этот период разработаны и запущены в производство вычислительные машины «Урал-1», «Урал-4», «Урал-11», «Урал-14», БЭСМ-2, М - 4 0, «Минск-1», «Минск-2», «Минск-22», «Минск-32». В 1960 году под руководством В. М. Глушкова и Б. Н. Малиновского разработана первая полупроводниковая управляющая машина «Днепр».

Наконец, нельзя не отметить значительные события в сфере Программного обеспечения, а именно создание языков программирования высокого уровня: Фортра на (1956), Алгола (1958) и Кобола (1959).

Третье поколение (1963-1972)

ретье поколение ознаменовалось резким увеличением вычислительной Мощности ВМ, ставшим следствием больших успехов в области архитектуры, технолог и программного обеспечения. Основные технологические достижения связанны с переходом от дискретных полупроводниковых элементов к интегральным микросхемам и началом применения полупроводниковых запоминающих устройств начинающих вытеснять ЗУ на магнитных сердечниках. Существенные изменений произошли и в архитектуре ВМ. Это, прежде всего, микропрограммирование как эффективная техника построения устройств управления сложных процессо а также наступление эры конвейеризации и параллельной обработки. В области программного обеспечения определяющими вехами стали первые операционные системы и реализация режима разделения времени.

В первых ВМ третьего поколения использовались интегральные схемы с малой степенью интеграции (small-scale integrated circuits, SSI), где на одном кр талле размещается порядка 10 транзисторов. Ближе к концу рассматриваемого периода на смену SSI стали приходить интегральные схемы средней степени интеграции (medium-scale integrated circuits, MSI), в которых число транзисторов на Кристалле увеличилось на порядок. К этому же времени относится повсеместное применение многослойных печатных плат. Все шире востребуются преимущества параллельной обработки; реализуемые за счет множественных функциональных блоков, совмещения во времени работы центрального процессор и операций ввода/вывода, конвейеризации потоков команд и данных.

В 1964 году Сеймур Крей (Seymour Cray, 1925-1996) построил вычислительную систему CDC 6600, в архитектуру которой впервые был заложен функциональный параллелизм. Благодаря наличию 10 независимых функциональных блоков, способных работать параллельно, и 32 независимых модулей памяти удалось достичь быстродействия в 1 MFLOPS (миллион операций с плавающей запятой в секунду). Пятью годами позже Крей создал CDC 7600 с конвейеризированными функциональными блоками и быстродействием 10 MFLOPS. CDC 7600 называют первой конвейерной вычислительной системой (конвейерным процессором). Революционной вехой в истории ВТ стало создание семейства вычислительных машин IBM 360, архитектура и программное обеспечение которых на долгие годы служили эталоном для последующих больших универсальных ВМ (mainframes). В машинах этого семейства нашли воплощение многие новые для того периода идеи, в частности: предварительная выборка команд, отдельные блоки для операций с фиксированной и плавающей запятой, конвейеризация команд, кэш-память. К третьему поколению ВС относятся также первые параллельные вычислительные системы: SOLOMON корпорации Westinghause и ILLIAC IV - совместная разработка Иллинойского университета и компании Burroughs. Третье поколение ВТ ознаменовалось также появлением первых конвейерно-векторных ВС: TI-ASC (Texas Instruments Advanced Scientific Computer) и STAR-100 фирмы CBC.

Среди вычислительных машин, разработанных в этот период в СССР, прежде всего необходимо отметить «быстродействующую электронно-счетную машину» - БЭСМ-6 (С. А. Лебедев) с производительностью 1 млн операций/с. Продолжением линии М-20 стали М-220 и М-222 с производительностью до 200 000 операций/с. Оригинальная ВМ для инженерных расчетов «Мир-1» была создана под руководством В. М. Глушкова. В качестве входного языка этой ВМ использован язык программирования высокого уровня «Аналитик», во многом напоминающий язык Алгол.

В сфере программного обеспечения необходимо отметить создание в 1970 год Кеном Томпсоном (Kenneth Thompson) из Bell Labs языка B, прямого предшественника популярного языка программирования C, и появление ранней версии операционной системы UNIX.

Четвертое поколение (1972-1984)

Отсчет четвертого поколения обычно ведут с перехода на интегральные микросхемы большой (large-scale integration, LSI) и сверхбольшой (very large-scale integration, VLSI) степени интеграции. К первым относят схемы, содержащие около 1000 транзисторов на кристалле, в то время как число транзисторов на одном кристалле VLSI имеет порядок 100 000. При таких уровнях интеграции стало возможным уместить в одну микросхему не только центральный процессор, но и вычислительную машину (ЦП, основную память и систему ввода/вывода).

Конец 70-х и начало 80-х годов — это время становления и последующего победного шествия микропроцессоров и микроЭВМ, что, однако, не снижает важности изменений, произошедших в архитектуре других типов вычислительных машин и систем.

Одним из наиболее значимых событий в области архитектуры ВМ стала идея вычислительной машины с сокращенным набором команд (RISC, Redused Instru

tion Set Computer), выдвинутая в 1975 году и впервые реализованная в 1980 году. В упрощенном изложении суть концепция RISC заключается в сведении набора команд ВМ к наиболее употребительным простейшим командам. Это позволяет упростить схемотехнику процессора и добиться резкого сокращения времени выполнения каждой из «простых» команд. Более сложные команды реализуются как подпрограммы, составленные из быстрых «простых» команд.

В ВМ и ВС четвертого поколения практически уходят со сцены ЗУ на магнитных сердечниках и основная память строится из полупроводниковых запоминающих устройств (ЗУ). До этого использование полупроводниковых ЗУ ограничивалось лишь регистрами и кэш-памятью.

В сфере высокопроизводительных вычислений доминируют векторные вычислительные системы, более известные как суперЭВМ. Разрабатываются новые параллельные архитектуры, однако подобные работы пока еще носят экспериментальный характер. На замену большим ВМ, работающим в режиме разделения времени, приходят индивидуальные микроЭВМ и рабочие станции (этим термином обозначают сетевой компьютер, использующий ресурсы сервера).

В области программного обеспечения выделим появление языков программирования сверхвысокого уровня, таких как FP (functional programming - функциональное программирование) и Пролог (Prolog, programming in logic). Эти языки ориентированы на *декларативный стиль программирования,* в отличие от Паскаля, С, Фортрана и т. д. — языков *императивного стиля программирования.* При декларативном стиле программист дает математическое описание того, что должно быть вычислено, а детали того, каким образом это должно быть сделано, возлагаются на компилятор и операционную систему. Такие языки пока используются недостаточно широко, но выглядят многообещающими для ВС с массовым параллелизмом, состоящими из более чем 1000 процессоров. Ъ компиляторах для ВС четвертого поколения начинают применяться сложные методы оптимизации кода.

Два события в области программного обеспечения связаны с Кеном Томпсоном (Kenneth Thompson) и Деннисом Ритчи (Dennis Ritchie) из Bell Labs. Это создание языка программирования С и его использование при написании операционной системы UNIX для машины DEC PDP-11. Такая форма написания операционной системы позволила быстро распространить UNIX на многие ВМ.

Пятое поколение (1984–1990)

Главным поводом для выделения вычислительных систем второй половины 80-х годов в самостоятельное поколение стало стремительное развитие ВС с сотнями процессоров, ставшее побудительным мотивом для прогресса в области параллельных вычислений. Ранее параллелизм вычислений выражался лишь в виде конвейеризации, векторной обработки и распределения работы между небольшим числом процессоров. Вычислительные системы пятого поколения обеспечивают такое распределение задач по множеству процессоров, при котором каждый из процессоров может выполнять задачу отдельного пользователя.

В рамках пятого поколения в архитектуре вычислительных систем сформировались два принципиально различных подхода: архитектура с совместно используемой памятью и архитектура с распределенной памятью.

Характерным примером первого подхода может служить система Sequent Balance 8000, в которой имеется большая основная память, разделяемая 20 процессорами. Помимо этого, каждый процессор оснащен собственной кэш-памятью. Каждый из процессоров может выполнять задачу своего пользователя, но при этом в составе программного обеспечения имеется библиотека подпрограмм, позволяющая программисту привлекать для решения своей задачи более одного процессора. Система широко использовалась для исследования параллельных алгоритмов и техники программирования.

Второе направление развития систем пятого поколения — системы с распределенной памятью, где каждый процессор обладает своим модулем памяти, а связь между процессорами обеспечивается сетью взаимосвязей. Примером такой ВС может служить система iPSC-1 фирмы Intel, более известная как «гиперкуб». Максимальный вариант системы включал 128 процессоров. Применение распределенной памяти позволило устранить ограничения в пропускной способности тракта «процессор-память», но потенциальным «узким местом» здесь становится сеть взаимосвязей.

Наконец, третье направление в архитектуре вычислительных систем пятого поколения — это ВС, в которых несколько тысяч достаточно простых процессоров работают под управлением единого устройства управления и одновременно производят одну и ту же операцию, но каждый над своими данными. К этому классу можно отнести Connection Machine фирмы Thinking Machines Inc. и MP-1 фирмы MasPar Inc.

В научных вычислениях по-прежнему ведущую роль играют векторные супер-ЭВМ. Многие производители предлагают более эффективные варианты с несколькими векторными процессорами, но число таких процессоров обычно невелико (от 2 до 8).

RISC-архитектура выходит из стадии экспериментов и становится базовой архитектурой для рабочих станций (workstations).

Знаковой приметой рассматриваемого периода стало стремительное развитие технологий глобальных и локальных компьютерных сетей. Это стимулировало изменения в технологии работы индивидуальных пользователей. В противовес мощным универсальным ВС, работающим в режиме разделения времени, пользователи все более отдают предпочтение подключенным к сети индивидуальным рабочим станциям. Такой подход позволяет для решения небольших задач задействовать индивидуальную машину, а при необходимости в большой вычислительной мощности обратиться к ресурсам подсоединенных к той же сети мощных файл-серверов или суперЭВМ.

Шестое поколение (1990-)

На ранних стадиях эволюции вычислительных средств смена поколений ассоциировалась с революционными технологическими прорывами. Каждое из первых четырех поколений имело четко выраженные отличительные признаки и вполне определенные хронологические рамки. Последующее деление на поколения уже не столь очевидно и может быть понятно лишь при ретроспективном взгляде на развитие вычислительной техники. Пятое и шестое поколения в эволюции ВТ — это отражение нового качества, возникшего в результате последовательного на-

копления частных достижений, главным образом в архитектуре вычислительных систем и, в несколько меньшей мере, в сфере технологий.

Поводом для начала отсчета нового поколения стали значительные успехи в области параллельных вычислений, связанные с широким распространением вычислительных систем с массовым параллелизмом. Особенности организации таких систем, обозначаемых аббревиатурой MPP (massively parallel processing), будут рассмотрены в последующих разделах. Здесь же упрощенно определим их как совокупность большого количества (до нескольких тысяч) взаимодействующих, но достаточно автономных вычислительных машин. По вычислительной мощности такие системы уже успешно конкурируют с суперЭВМ, которые, как ранее отмечалось, по своей сути являются векторными ВС. Появление вычислительных систем с массовым параллелизмом дало основание говорить о производительности, измеряемой в TFLOPS (1 TFLOPS соответствует 10^{12} операциям с плавающей запятой в секунду).

Вторая характерная черта шестого поколения — резко возросший уровень рабочих станций. В процессорах новых рабочих станций успешно совмещаются RISC-архитектура, конвейеризация и параллельная обработка. Некоторые рабочие станции по производительности сопоставимы с суперЭВМ четвертого поколения. Впечатляющие характеристики рабочих станций породили интерес к гетерогенным (неоднородным) вычислениям, когда программа, запущенная на одной рабочей станции, может найти в локальной сети не занятые в данный момент другие станции, после чего вычисления распараллеливаются и на эти простаивающие станции.

Наконец, третьей приметой шестого поколения в эволюции ВТ стал взрывной рост глобальных сетей. Этот момент, однако, выходит за рамки данной книги, поэтому далее комментироваться не будет.

Завершая обсуждение эволюции ВТ, отметим, что верхняя граница шестого поколения хронологически пока не определена и дальнейшее развитие вычислительной техники может внести в его характеристику новые коррективы. Не исключено также, что последующие события дадут повод говорить и об очередном поколении.

Концепция машины с хранимой в памяти программой

Исходя из целей данного раздела, введем новое определение термина «вычислительная машина» как совокупности технических средств, служащих для автоматизированной обработки дискретных данных по заданному алгоритму.

Алгоритм - одно из фундаментальных понятий математики и вычислительной техники. Международная организация стандартов (ISO) формулирует понятие *алгоритм* как «конечный набор предписаний, определяющий решение задачи посредством конечного количества операций» (ISO 2382/1-84). Помимо этой стандартизированной формулировки существуют и другие определения. Приведем наиболее распространенные из них. Итак, алгоритм — это:

- способ преобразования информации, задаваемый с помощью конечной системы правил;

- совокупность правил, определяющих эффективную процедуру решения любой задачи из некоторого заданного класса задач;
- точно определенное правило действий, для которого задано указание, как и в какой последовательности это правило необходимо применять к исходным данным задачи, чтобы получить ее решение.

Основными свойствами алгоритма являются: дискретность, определенность, массовость и результативность.

Дискретность выражается в том, что алгоритм описывает действия над дискретной информацией (например, числовой или символьной), причем сами эти действия также дискретны.

Свойство *определенности* означает, что в алгоритме указано все, что должно быть сделано, причем ни одно из действий не должно трактоваться двояко.

Массовость алгоритма подразумевает его применимость к множеству значений исходных данных, а не только к каким-то уникальным значениям.

Наконец, *результативность* алгоритма состоит в возможности получения результата за конечное число шагов.

Рассмотренные свойства алгоритмов предопределяют возможность их реализации на ВМ, при этом процесс, порождаемый алгоритмом, называют *вычислительным процессом*.

В основе архитектуры современных ВМ лежит представление алгоритма решения задачи в виде программы последовательных вычислений. Согласно стандарту ISO 2382/1-84, программа для ВМ — это «упорядоченная последовательность команд, подлежащая обработке».

ВМ, где определенным образом закодированные команды программы хранятся в памяти, известна под названием *вычислительной машины с хранимой в памяти программой*. Идея принадлежит создателям вычислителя ENIAC Эккерту, Мочли и фон Нейману. Еще до завершения работ над ENIAC они приступили к новому проекту - EDVAC, главной особенностью которого стала концепция хранимой в памяти программы, на долгие годы определившая базовые принципы построения последующих поколений вычислительных машин. Относительно авторства существует несколько версий, но поскольку в законченном виде идея впервые была изложена в 1945 году в статье фон Неймана [219], именно его фамилия фигурирует в обозначении архитектуры подобных машин, составляющих подавляющую часть современного парка ВМ и ВС.

Сущность фон-неймановской концепции вычислительной машины можно свести к четырем принципам:

- двоичного кодирования;
- программного управления;
- однородности памяти;
- адресности.

Принцип двоичного кодирования

Согласно этому принципу, вся информация, как данные, так и команды, кодируются двоичными цифрами 0 и 1. Каждый тип информации представляется двоич-

ной последовательностью и имеет свой *формат*. Последовательность битов в формате, имеющая определенный смысл, называется *полем*. В числовой информации обычно выделяют *поле знака* и *поле значащих разрядов*. В формате команды можно выделить два поля (рис. 1.2): *поле кода операции* (КОп) и *поле адресов* (адресную часть — АЧ).

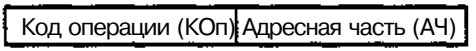


Рис. 1.2. Структура команды

Код операции представляет собой указание, какая операция должна быть выполнена, и задается с помощью r-разрядной двоичной комбинации.

Вид адресной части и число составляющих ее адресов зависят от типа команды: в командах преобразования данных АЧ содержит адреса объектов обработки *(операндов)* и результата; в командах изменения порядка вычислений — адрес следующей команды программы; в командах ввода/вывода - номер устройства ввода/вывода. Адресная часть также представляется двоичной последовательностью, длину которой обозначим через/?. Таким образом, команда в вычислительной машине имеет вид $(r + p)$-разрядной двоичной комбинации.

Принцип программного управления

Все вычисления, предусмотренные алгоритмом решения задачи, должны быть представлены в виде *программы,* состоящей из последовательности управляющих слов - *команд*. Каждая команда предписывает некоторую операцию из набора операций, реализуемых вычислительной машиной. Команды программы хранятся в последовательных ячейках памяти вычислительной машины и выполняются *в естественной последовательности,* то есть в порядке их положения в программе. При необходимости, с помощью специальных команд, эта последовательность может быть изменена. Решение об изменении порядка выполнения команд программы принимается либо на основании анализа результатов предшествующих вычислений, либо безусловно.

Принцип однородности памяти

Команды и данные хранятся в одной и той же памяти и внешне в памяти неразличимы. Распознать их можно только по способу использования. Это позволяет производить над командами те же операции, что и над числами, и, соответственно, открывает ряд возможностей. Так, циклически изменяя адресную часть команды, можно обеспечить обращение к последовательным элементам массива данных. Такой прием носит название *модификации команд* и с позиций современного программирования не приветствуется. Более полезным является другое следствие принципа однородности, когда команды одной программы могут быть получены как результат исполнения другой программы. Эта возможность лежит в основе *трансляции* — перевода текста программы с языка высокого уровня на язык конкретной ВМ.

Концепция вычислительной машины, изложенная в статье фон Неймана, предполагает единую память для хранения команд и данных. Такой подход был принят

в вычислительных машинах, создававшихся в Принстонском университете, из-за чего и получил название *принстонской архитектуры*. Практически одновременно в Гарвардском университете предложили иную модель, в которой ВМ имела отдельную память команд и отдельную память данных. Этот вид архитектуры называют *гарвардской архитектурой*. Долгие годы преобладающей была и остается принстонская архитектура, хотя она порождает проблемы пропускной способности тракта «процессор-память». В последнее время в связи с широким использованием кэш-памяти разработчики ВМ все чаще обращаются к гарвардской архитектуре.

Принцип адресности

Структурно основная память состоит из пронумерованных ячеек, причем процессору в произвольный момент доступна любая ячейка. Двоичные коды команд и данных разделяются на единицы информации, называемые *словами,* и хранятся в ячейках памяти, а для доступа к ним используются номера соответствующих ячеек — *адреса*. .

Фон-неймановская архитектура

В статье фон Неймана определены основные устройства ВМ, с помощью которых должны быть реализованы вышеперечисленные принципы. Большинство современных ВМ по своей структуре отвечают принципу программного управления. Типичная фон-неймановская ВМ (рис. 1.3) содержит: память, устройство управления, арифметико-логическое устройство и устройство ввода/вывода.

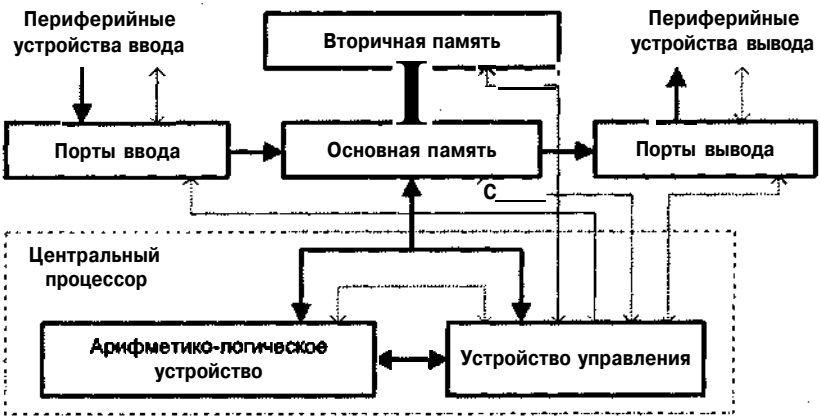


Рис. 1.3.Структура фон-неймановской вычислительной машины

В любой ВМ имеются средства для ввода программ и данных к ним. Информация поступает из подсоединенных к ЭВМ *периферийных устройств* (ПУ) ввода. Результаты вычислений выводятся на периферийные устройства вывода. Связь и взаимодействие ВМ и ПУ обеспечивают *порты ввода* и *порты вывода*. Термином *порт* обозначают аппаратуру сопряжения периферийного устройства с ВМ и управления им. Совокупность портов ввода и вывода называют *устройством ввода/вывода* (УВВ) или *модулем ввода/вывода* ВМ (МВБ).

Введенная информация сначала запоминается в основной памяти, а затем переносится во вторичную память, для длительного хранения. Чтобы программа могла выполняться, команды и данные должны располагаться в *основной памяти* (ОП), организованной таким образом, что каждое двоичное слово хранится в отдельной ячейке, идентифицируемой адресом, причем соседние ячейки памяти имеют следующие по порядку адреса. Доступ к любым ячейкам запоминающего устройства (ЗУ) основной памяти может производиться в произвольной последовательности. Такой вид памяти известен как *память с произвольным доступом*. ОП современных ВМ в основном состоит из полупроводниковых *оперативных запоминающих устройств* (ОЗУ), обеспечивающих как считывание, так и запись информации. Для таких ЗУ характерна энергозависимость — хранимая информация теряется при отключении электропитания. Если необходимо, чтобы часть основной памяти была энергонезависимой, в состав ОП включают *постоянные запоминающие устройства* (ПЗУ), также обеспечивающие произвольный доступ. Хранящаяся в ПЗУ информация может только считываться (но не записываться).

Размер ячейки основной памяти обычно принимается равным 8 двоичным разрядам — *байту*. Для хранения больших чисел используются 2, 4 или 8 байтов, размещаемых в ячейках с последовательными адресами. В этом случае за адрес числа часто принимается адрес его младшего байта. Так, при хранении 32-разрядного числа в ячейках с адресами 200, 201, 202 и 203 адресом числа будет 200. Такой прием называют адресацией по младшему байту или методом «остроконечников» *(little endian addressing)*. Возможен и противоположный подход - по меньшему из адресов располагается старший байт. Этот способ известен как адресация по старшему байту или метод «тупоконечников» *(big endian addressing)*[1]. Адресация по младшему байту характерна для микропроцессоров фирмы Intel и мини-ЭВМ фирмы DEC, а по старшему байту — для микропроцессоров фирмы Motorola и универсальных ЭВМ фирмы IBM. В принципе выбор порядка записи байтов существенен лишь при пересылке данных между ВМ с различными формами их адресации или при манипуляциях с отдельными байтами числа. В большинстве ВМ предусмотрены специальные инструкции для перехода от одного способа к другому.

Для долговременного хранения больших программ и массивов данных в ВМ обычно имеется дополнительная память, известная как *вторичная*. Вторичная память энергонезависима и чаще всего реализуется на базе магнитных дисков. Информация в ней хранится в виде специальных программно поддерживаемых объектов - *файлов* (согласно стандарту ISO, файл - это «идентифицированная совокупность экземпляров полностью описанного в конкретной программе типа данных, находящихся вне программы во внешней памяти и доступных программе посредством специальных операций»).

Устройство управления (УУ) — важнейшая часть ВМ, организующая автоматическое выполнение программ (путем реализации функций управления) и обеспечивающая функционирование ВМ как единой системы. Для пояснения функций УУ ВМ следует рассматривать как совокупность элементов, между которыми

[1] Термины «остроконечники» и «тупоконечники» заимствованы из книги «Путешествия Гулливера» Дж. Свифта, где упоминается религиозная война между двумя группами, представители одной из которых разбивали яйцо с острого (little) конца, а их антагонисты — с тупого (big).

происходит пересылка информации, в ходе которой эта информация может подвергаться определенным видам обработки. Пересылка информации между любыми элементами ВМ инициируется своим *сигналом управления* (СУ), то есть управление вычислительным процессом сводится к выдаче нужного набора СУ в нужной временной последовательности. Цепи СУ показаны на рис. 13 полутоновыми линиями. Основной функцией УУ является формирование управляющих сигналов, отвечающих за извлечение команд из памяти в порядке, определяемом программой, и последующее исполнение этих команд. Кроме того, УУ формирует СУ для синхронизации и координации внутренних и внешних устройств ВМ.

Еще одной неотъемлемой частью ВМ является *арифметико-логическое устройство* (АЛУ). АЛУ обеспечивает арифметическую и логическую обработку двух входных переменных, в результате которой формируется выходная переменная. Функции АЛУ обычно сводятся к простым арифметическим и логическим операциям, а также операциям сдвига. Помимо результата операции АЛУ формирует ряд *признаков результата* (флагов), характеризующих полученный результат и события, произошедшие в процессе его получения (равенство нулю, знак, четность, перенос, переполнение и т. д.). Флаги могут анализироваться в УУ с целью принятия решения о дальнейшей последовательности выполнения команд программы.

УУ и АЛУ тесно взаимосвязаны и их обычно рассматривают как единое устройство, известное *как центральный процессор* (ЦП) или просто *процессор*. Помимо УУ и АЛУ в процессор входит также набор *регистров общего назначения* (РОН), служащих для промежуточного хранения информации в процессе ее обработки.

Типы структур вычислительных машин и систем

Достоинства и недостатки архитектуры вычислительных машин и систем изначально зависят от способа соединения компонентов. При самом общем подходе можно говорить о двух основных типах структур вычислительных машин и двух типах структур вычислительных систем.

Структуры вычислительных машин .

В настоящее время примерно одинаковое распространение получили два способа построения вычислительных машин: *с непосредственными связями* и *на основе шины*. .

Типичным представителем первого способа может служить классическая фон-неймановская ВМ (см. рис. 1.3). В ней между взаимодействующими устройствами (процессор, память, устройство ввода/вывода) имеются непосредственные связи. Особенности связей (число линий в шинах, пропускная способность и т. п.) определяются видом информации, характером и интенсивностью обмена Достоинством архитектуры с непосредственными связями можно считать возможность развязки «узких мест» путем улучшения структуры и характеристик только определенных связей, что экономически может быть наиболее выгодным решением. У фон-неймановских ВМ таким «узким местом» является канал пересылки данных между ЦП и памятью, и «развязать» его достаточно непросто [56]. Кроме того, ВМ с непосредственными связями плохо поддаются реконфигурации.

В варианте с общей шиной все устройства вычислительной машины подключены к магистральной шине, служащей единственным трактом для потоков команд, данных и управления (рис. 1.4). Наличие общей шины существенно упрощает реализацию ВМ, позволяет легко менять состав и конфигурацию машины. Благодаря этим свойствам шинная архитектура получила широкое распространение в мини- и микроЭВМ. Вместе с тем, именно с шиной связан и основной недостаток архитектуры: в каждый момент передавать информацию по шине может только одно устройство. Основную нагрузку на шину создают обмены между процессором и памятью, связанные с извлечением из памяти команд и данных и записью в память результатов вычислений. На операции ввода/вывода остается лишь часть пропускной способности шины. Практика показывает, что даже при достаточно быстрой шине для 90% приложений этих остаточных ресурсов обычно не хватает, особенно в случае ввода или вывода больших массивов данных.

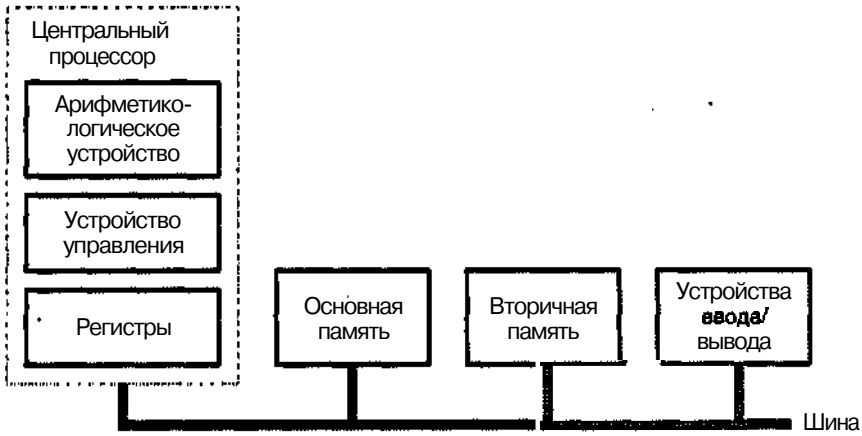


Рис. 1.4. Структура вычислительной машины на базе общей шины

В целом следует признать, что при сохранении фон-неймановской концепции последовательного выполнения команд программы шинная архитектура в чистом ее виде оказывается недостаточно эффективной. Более распространена *архитектура с иерархией шин,* где помимо магистральной шины имеется еще несколько дополнительных шин. Они могут обеспечивать непосредственную связь между устройствами с наиболее интенсивным обменом, например процессором и кэш-памятью. Другой вариант использования дополнительных шин — объединение однотипных устройств ввода/вывода с последующим выходом с дополнительной шины на магистральную. Все эти меры позволяют снизить нагрузку на общую шину и более эффективно расходовать ее пропускную способность.

Структуры вычислительных систем

Понятие «вычислительная система» предполагает наличие множества процессоров или законченных вычислительных машин, при объединении которых используется один из двух подходов.

В вычислительных системах с общей памятью (рис. 1.5) имеется общая основная память, совместно используемая всеми процессорами системы. Связь процес-

соров с памятью обеспечивается с помощью коммуникационной сети, чаще всего вырождающейся в общую шину. Таким образом, структура ВС с общей памятью аналогична рассмотренной выше архитектуре с общей шиной, в силу чего ей свойственны те же недостатки. Применительно к вычислительным системам данная схема имеет дополнительное достоинство: обмен информацией между процессорами не связан с дополнительными операциями и обеспечивается за счет доступа к общим областям памяти.

Рис 1.5. Структура вычислительной системы с общей памятью

Альтернативный вариант организации - *распределенная* система, где общая память вообще отсутствует, а каждый процессор обладает собственной локальной памятью (рис. 1.6). Часто такие системы объединяют отдельные ВМ. Обмен информацией между составляющими системы обеспечивается с помощью коммуникационной сети посредством обмена сообщениями.

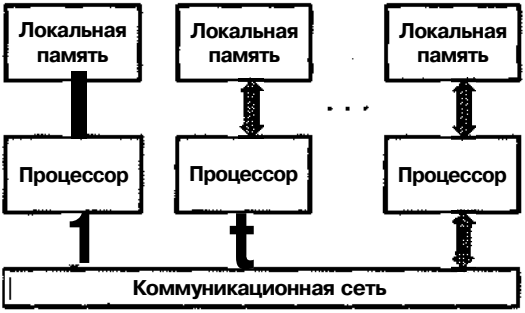


Рис 1.6. Структура распределенной вычислительной системы

Подобное построение ВС снимает ограничения, свойственные для общей шины, но приводит к дополнительным издержкам на пересылку сообщений между процессорами или машинами.

Перспективы совершенствования архитектуры ВМ и ВС

Совершенствование архитектуры вычислительных машин и систем началось с момента появления первых ВМ и не прекращается по сей день. Каждое изменение в архитектуре направлено на абсолютное повышение производительности или, по крайней мере, на более эффективное решение задач определенного класса. Эволюцию архитектур определяют самые различные факторы, главные из которых показаны на рис. 1.7. Не умаляя роли ни одного из них, следует признать, что наиболее

очевидные успехи в области средств вычислительной техники все же связаны с технологическими достижениями. Характер и степень влияния прочих факторов подробно описаны в [120] и в данном учебнике не рассматриваются.

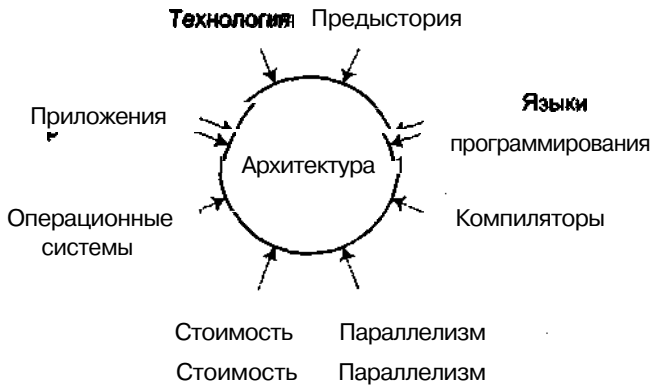


Рис. **1.7.** Факторы, определяющие развитие архитектуры вычислительных систем

С каждым новым технологическим успехом многие из архитектурных идей переходят на уровень практической реализации. Очевидно, что процесс этот будет продолжаться и в дальнейшем, однако возникает вопрос: «Насколько быстро?» Косвенный ответ можно получить, проанализировав тенденции совершенствования технологий, главным образом полупроводниковых.

Тенденции развития больших интегральных схем

На современном уровне вычислительной техники подавляющее большинство устройств ВМ и ВС реализуется на базе полупроводниковых технологий в виде *сверхбольших интегральных микросхем* (СБИС). Каждое нововведение в области архитектуры ВМ и ВС, как правило, связано с необходимостью усложнения схемы процессора или его составляющих и требует размещения на кристалле СБИС все большего числа логических или запоминающих элементов. Задача может быть решена двумя путями: увеличением размеров кристалла и уменьшением площади, занимаемой на кристалле элементарным транзистором, с одновременным повышением плотности упаковки таких транзисторов на кристалле.

Наиболее перспективным представляется увеличение размеров кристалла, однако только на первый взгляд. Кристаллической подложкой микросхемы служит тонкая пластина, представляющая собой срез цилиндрического бруска полупроводникового материала. Полезная площадь подложки ограничена вписанным в окружность квадратом или прямоугольником. Увеличение диаметра кристаллической подложки на 10% на практике позволяет получить до 60% прироста числа транзисторов на кристалле. К сожалению, технологические сложности, связанные с изготовлением кристаллической подложки большого размера без ухудшения однородности ее свойств по всей поверхности, чрезвычайно велики. Фактические тенденции в плане увеличения размеров кристаллической подложки СБИС иллюстрирует рис. 1.8.

Точки излома на графике соответствуют годам, когда переход на новый размер кристалла становится повсеместным. Каждому переходу обычно предше-

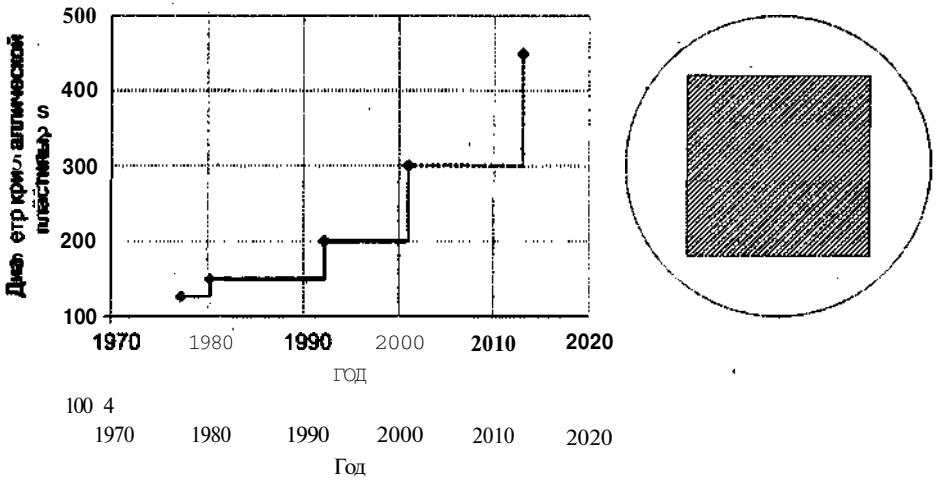


Рис. 1.8. Тенденции увеличения диаметра кристаллической подложки СБИС

ствуют 2-3-летние исследования, а собственно переход на пластины увеличенного диаметра происходит в среднем один раз в 9 лет.

Пока основные успехи в плане увеличения емкости СБИС связаны с уменьшением размеров элементарных транзисторов и плотности их размещения на кристалле. Здесь тенденции эволюции СБИС хорошо описываются эмпирическим законом Мура [168]. В 1965 году Мур заметил, что число транзисторов, которое удается разместить на кристалле микросхемы, удваивается каждые 12 месяцев. Он предсказал, что эта тенденция сохранится в 70-е годы, а начиная с 80-х темп роста начнет спадать. В 1995 году Мур уточнил свое предсказание, сделав прогноз, что удвоение числа транзисторов далее будет происходить каждые 24 месяца.

Создание интегральных микросхем предполагает два этапа. Первый из них носит название *литографии* и заключается в получении маски, определяющей структуру будущей микросхемы. На втором этапе маска накладывается на полупроводниковую пластину, после чего пластина облучается, в результате чего и формируется микросхема. Уменьшение размеров элементов на кристалле напрямую зависит от возможностей технологии (рис. 19).

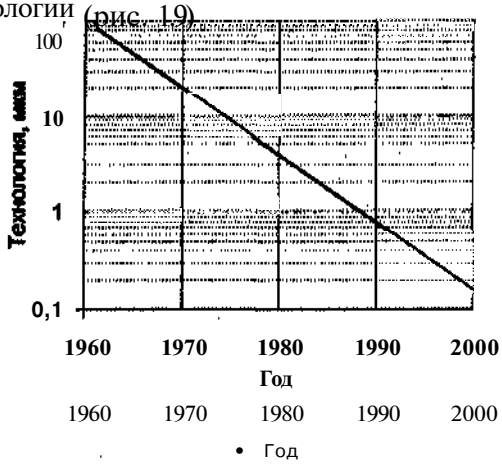


Рис 1.9. Размер минимального элемента на кристалле интегральной микросхемы

Современный уровень литографии сделал возможным серийный выпуск СБИС, в которых размер элемента не превышает 0,13 мкм. Чтобы оценить перспективы развития возможностей литографии на ближайший период, обратимся к прогнозу авторитетного эксперта в области полупроводниковых технологий - International Technology Roadmap for Semiconductors. Результаты прогноза относительно будущих достижений литографии, взятые из отчета за 2001 год [185], приведены на рис. 1.10.

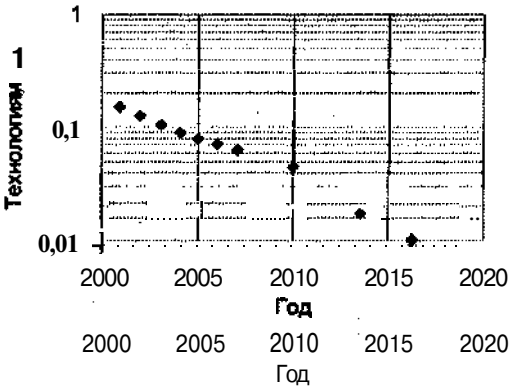


Рис. **1.10.** Прогноз максимальных размеров элементов на кристалле **СБИС**

Наконец, еще одна общая тенденция в технологии СБИС - переход от алюминиевых соединительных линий на кристалле на медные. «Медная» технология позволяет повысить быстродействие СБИС примерно на 10% с одновременным снижением потребляемой мощности.

Приведенные выше закономерности определяют общие направления совершенствования технологий СБИС. Для более объективного анализа необходимо принимать во внимание функциональное назначение микросхем. В аспекте архитектуры ВМ и ВС следует отдельно рассмотреть «процессорные» СБИС и СБИС запоминающих устройств.

Тенденции развития элементной базы процессорных устройств

Современные технологии производства сверхбольших интегральных микросхем позволяют разместить на одном кристалле логические схемы всех компонентов процессора. В настоящее время процессоры всех вычислительных машин реализуются в виде одной или нескольких СБИС. Более того, во многих многопроцессорных ВС используются СБИС, где на одном кристалле располагаются сразу несколько процессоров (обычно не очень сложных). Каждый успех создателей процессорных СБИС немедленно положительно отражается на характеристиках ВМ и ВС. Совершенствование процессорных СБИС ведется по разным направлениям. Для целей данного учебника основной интерес представляет увеличение количества логических элементов, которое может быть размещено на кристалле, и повышение быстродействия этих логических элементов. Увеличение быстродействия ведет к наращиванию производительности процессоров даже без изменения их архитектуры, а в совокупности с повышением плотности упаковки логических эле-

ментов открывает возможности для реализации ранее недоступных архитектурных решений.

К увеличению числа логических элементов на кристалле ведут три пути:

- увеличение размеров кристалла;
- уменьшение размеров элементарных транзисторов;
- уменьшение ширины проводников, образующих внутренние шины или соединяющих логические элементы между собой.

Увеличение размеров кристаллов процессорных СБИС происходит в соответствии с ранее рассмотренными общими тенденциями и не имеет каких-либо особенностей.

Плотность упаковки логических элементов в процессорных СБИС принято оценивать количеством транзисторов, из которых, собственно, и строятся логические схемы процессора. Общие тенденции в плане плотности упаковки проследим на примере линейки микропроцессоров фирмы Intel (рис. 1.11). Из рисунка видно, что количество транзисторов в микропроцессорах, выпущенных до 2002 года, хорошо согласуется с законом Мура. Та же закономерность прослеживается и для других типов процессорных СБИС. Достаточно близки и абсолютные показатели разных микропроцессоров, выпущенных приблизительно в один и тот же период. Так, микропроцессор Pentium 4 фирмы Intel содержит 42 млн транзисторов, а микропроцессор Athlon XL фирмы AMD - 37 млн.

Чтобы оценить перспективы роста плотности упаковки на ближайшие два десятилетия, на рис. 1.11 дополнительно приведены прогностические данные на пе-

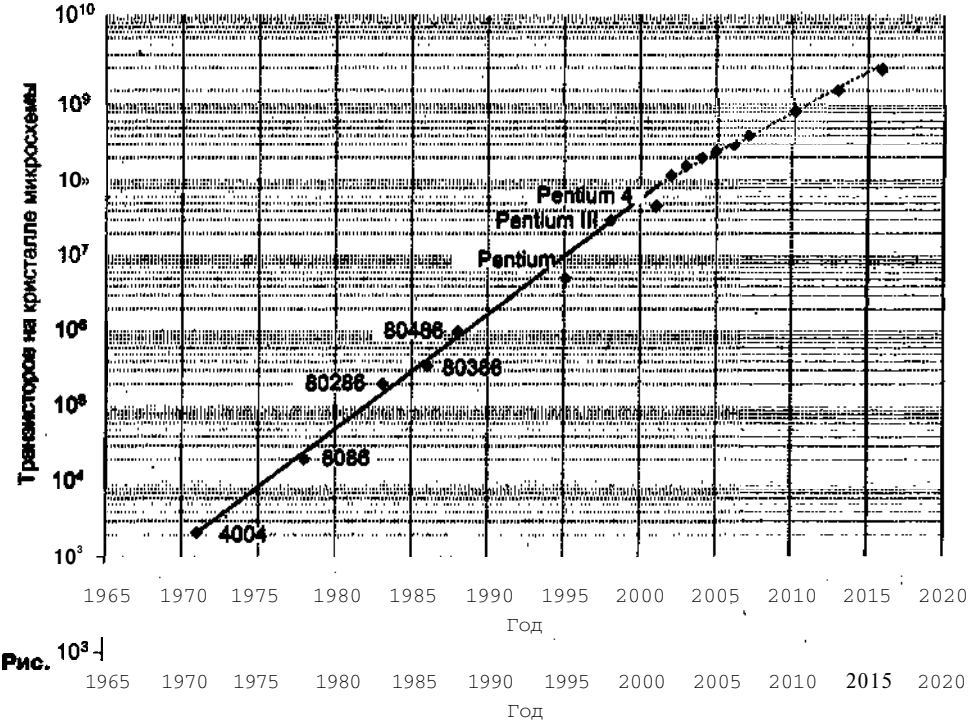


Рис. 1.11. Тенденции увеличения количества транзисторов на кристаллах процессорных СБИС

риод до 2020 года, взятые из [185]. Нетрудно заметить, что прогноз также не слишком расходится с уточненным законом Мура. Общий итог можно сформулировать следующим образом: *плотность упаковки логических схем процессорных СБИС каждые два года будет возрастать вдвое.*

В качестве параметра, характеризующего быстродействие логических схем процессорных СБИС, обычно используют так называемую внутреннюю тактовую частоту. На рис. 1.12 показаны значения тактовых частот микропроцессоров фирмы Intel. Из графика видно стремление к росту внутренней тактовой частоты процессорных СБИС: *удвоение частоты происходит в среднем каждые два года.* На рисунке присутствует также прогноз на ближайший период (данные взяты из [185]), из которого явствует, что в ближайшем будущем темп увеличения внутренней тактовой частоты может несколько снизиться.

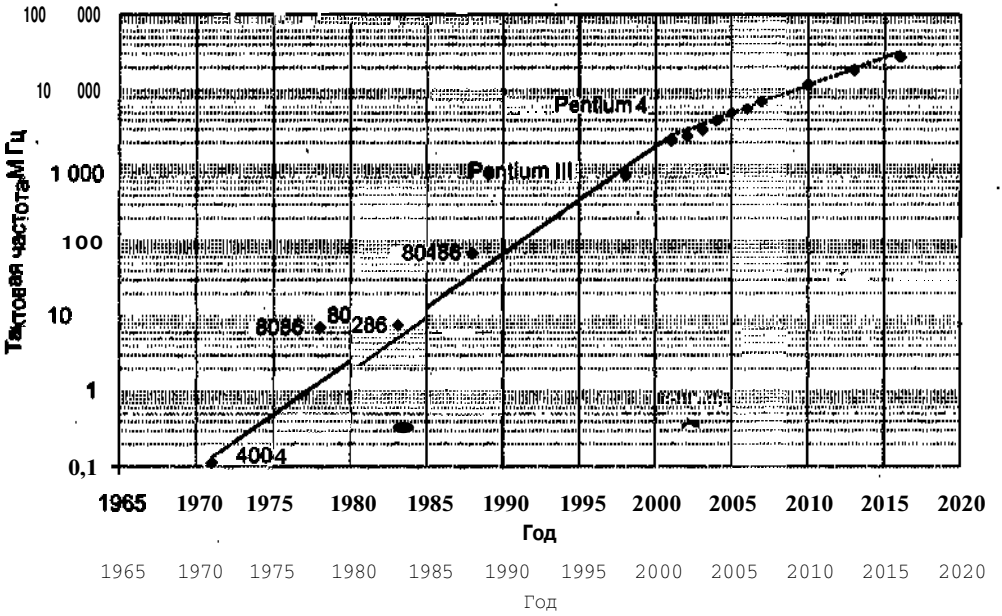


Рис. 1.12. Тенденции увеличения внутренней тактовой частоты процессорных СБИС

Тенденции развития полупроводниковых запоминающих устройств

По мере повышения возможностей вычислительных средств растут и «аппетиты» программных приложений относительно емкости основной памяти. Эту ситуацию отражает так называемый закон Паркинсона: «Программное обеспечение увеличивается в размерах до тех пор, пока не заполнит всю доступную на данный момент память». В цифрах тенденция возрастания требований к емкости памяти выглядит так: увеличение в полтора раза каждые два года. Основная память современных ВМ и ВС формируется из СБИС полупроводниковых запоминающих устройств, главным образом динамических ОЗУ. Естественные требования к таким СБИС: высокие плотность упаковки запоминающих элементов и быстродействие, низкая стоимость.

Плотность упаковки запоминающих элементов на кристалле динамического ОЗУ принято характеризовать емкостью хранимой информации в битах. Представление о сойременном состоянии и перспективах на ближайшее будущее дает график, приведенный на рис. 1.13. Для СБИС памяти также подтверждается справедливость закона Мураи предсказанное им уменьшение темп'ов повышения плотности упаковки. В целом можно предсказать, что *число запоминающих элементов на кристалле будет возрастать в два раза каждые полтора года.*

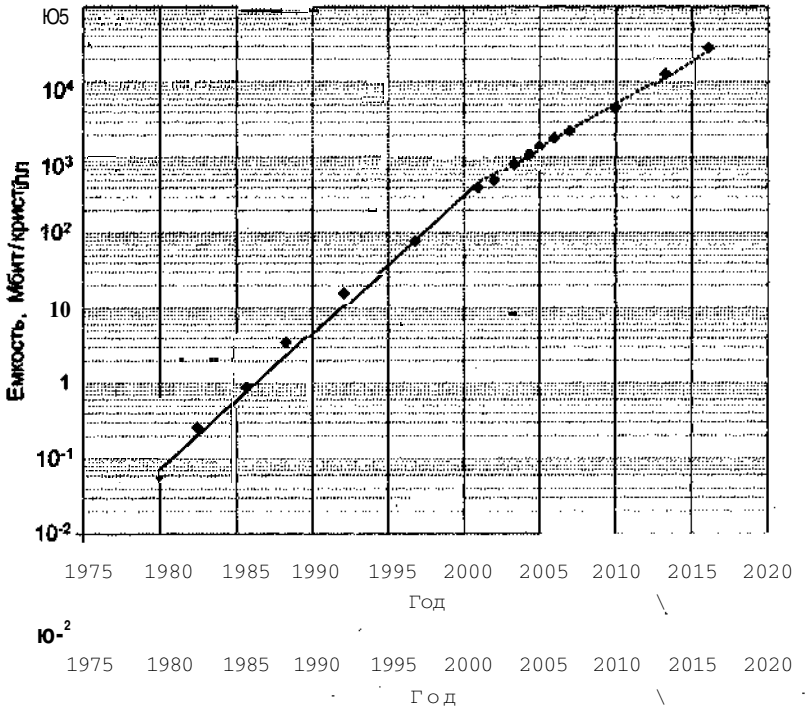


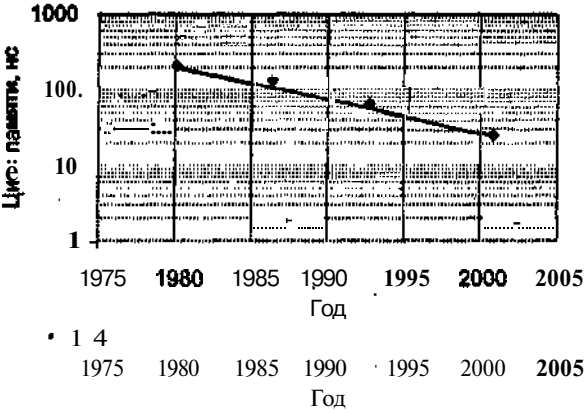
Рис. 1.13. Тенденции увеличения количества запоминающих элементов на кристалле

Рис. 1.14. Разрыв в производительности процессоров и динамических запоминающих устройств

С быстродействием СБИС памяти дело обстоит хуже. Высокая скорость процессоров уже давно находится в противоречии с относительной медлительностью

запоминающих устройств основной памяти. Проблема постоянно усугубляется несоответствием темпов роста тактовой частоты процессоров и быстродействия памяти, и особых перспектив в этом плане пока не видно, что иллюстрирует рис. 1.14.

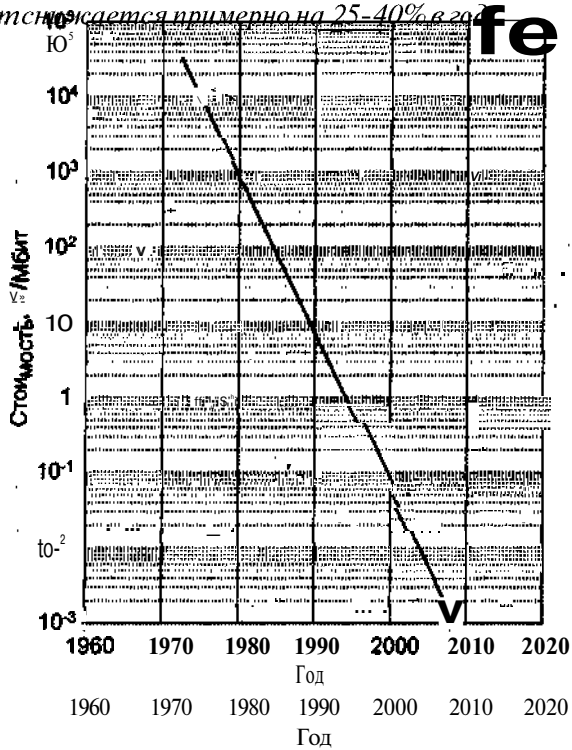
Абсолютные темпы снижения длительности цикла памяти, начиная с 1980 года, показаны на рис. 1.15. Общая тенденция: *на двукратное уменьшение длительности цикла динамического ЗУ уходит примерно 15 лет.*



1 4
1975 1980 1985 1990 1995 2000 2005
Год

Рис. **1.15.** Быстродействие микросхем динамической памяти

В плане снижения стоимости СБИС памяти перспективы весьма обнадеживающие (рис. 1.16). В течение достаточно длительного времени *стоимость в пересчете на один бит снижается примерно на 25-40% в год.*



1960 1970 1980 1990 2000 2010 2020
Год

Рис. **1.16.** Тенденции снижения стоимости СБИС динамической памяти в пересчете на 1 Мбит

Перспективные направления исследований в области архитектуры

Основные направления исследований в области архитектуры ВМ и ВС можно условно разделить на две группы: эволюционные и революционные. К первой группе следует отнести исследования, целью которых является совершенствование методов реализации уже достаточно известных идей. Изыскания, условно названные революционными, направлены на создание совершенно новых архитектур, принципиально отличных от уже ставшей традиционной фон-неймановской архитектуры.

Большинство из исследований, относимых к эволюционным, связано с совершенствованием архитектуры микропроцессоров (МП). В принципе кардинально новых архитектурных подходов в микропроцессорах сравнительно мало. Основные идеи, лежащие в основе современных МП, были выдвинуты много лет тому назад, но из-за несовершенства технологии и высокой стоимости реализации нашли применение только в больших универсальных ВМ (мэйнфреймах) и супер-ЭВМ. Наиболее значимые из изменений в архитектуре МП связаны с повышением уровня параллелизма на уровне команд (возможности одйовременного выполнения нескольких команд). Здесь в первую очередь следует упомянуть конвейеризацию, суперскалярную обработку и архитектуру с командными словами сверхбольшой длины (VLIW). После успешного переноса на МП глобальных архитектурных подходов «больших» систем основные усилия исследователей теперь направлены на частные архитектурные изменения. Примерами таких эволюционных архитектурных изменений могут служить: усовершенствованные методы предсказания переходов в конвейере команд, повышение частоты успешных обращений к кэш-памяти за счет усложненных способов буферизации и т. п.

Наблюдаемые нами достижения в области вычислительных средств широкого применения пока обусловлены именно «эволюционными» исследованиями. Однако уже сейчас очевидно, что, оставаясь в рамках традиционных архитектур, мы довольно скоро натолкнемся на технологические ограничения. Один из путей преодоления технологического барьера лежит в области нетрадиционных подходов. Исследования, проводимые в этом направлении, по нашей классификации отнесены к «революционным». Справедливость такого утверждения подтверждается первыми образцами ВС с нетрадиционной архитектурой.

Оценивая перспективы эволюционного и революционного развития вычислительной техники, можно утверждать, что на ближайшее время наибольшего прогресса можно ожидать на пути использования идей параллелизма на всех его уровнях и создания эффективной иерархии запоминающих устройств.

Контрольные вопросы

1. По каким признакам можно разграничить понятия «вычислительная машина» и «вычислительная система»?
2. В чем состоит различие между «узкой» и «широкой» трактовкой понятия «архитектура вычислительной машины»?

3. Какой уровень детализации вычислительной машины позволяет определить, можно ли данную ВМ причислить к фон-неймановским?
4. Какие закономерности в эволюции вычислительных машин породили появление нового научного направления — «Теория эволюции компьютеров»?
5. По каким признакам выделяют поколения вычислительных машин?
6. Поясните определяющие идеи для каждого из этапов эволюции вычислительной техники.
7. Какой из принципов фон-неймановской концепции вычислительной машины можно рассматривать в качестве наиболее существенного?
8. Оцените достоинства и недостатки архитектур вычислительных машин с непосредственными связями и общей шиной.
9. Сформулируйте основные тенденции развития интегральной схемотехники.
10. Какие выводы можно сделать, исходя из закона Мура?
11. Охарактеризуйте основные направления в дальнейшем развитии архитектуры вычислительных машин и систем.

Глава 2
Архитектура системы команд

Системой команд вычислительной машины называют полный перечень команд, которые способна выполнять данная ВМ. В свою очередь, под *архитектурой системы команд* (АСК) принято определять те средства вычислительной машины, которые видны и доступны программисту. АСК можно рассматривать как линию согласования нужд разработчиков программного обеспечения с возможностями создателей аппаратуры вычислительной машины (рис. 2.1).

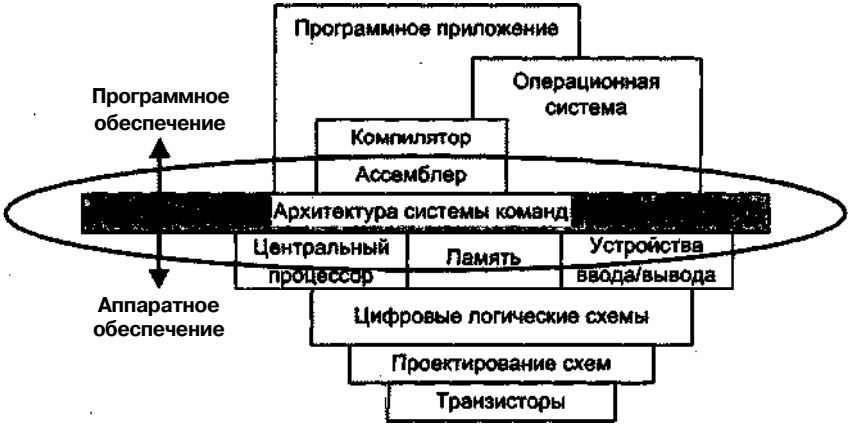


Рис. **2.1.** Архитектурасистемы команд как интерфейс меж
и аппаратным обеспечением

В конечном итоге, цель тех и других — реализация вычислений наиболее эффективным образом, то есть за минимальное время, и здесь важнейшую роль играет правильный выбор архитектуры системы команд.

В упрощенной трактовке время выполнения программы (Tвыч) можно определить через число команд в программе ($N_{ком}$), среднее количество тактов процессора, приходящихся на одну команду *(CPI),* и длительность тактового периода:

$$T_{выч} = N_{ком} \times CPI \times \tau_{др}.$$

Каждая из составляющих выражения зависит от одних аспектов архитектуры системы команд и, в свою очередь, влияет на другие (рис. 2.2), что свидетельствует о необходимости чрезвычайно ответственного подхода к выбору АСК.

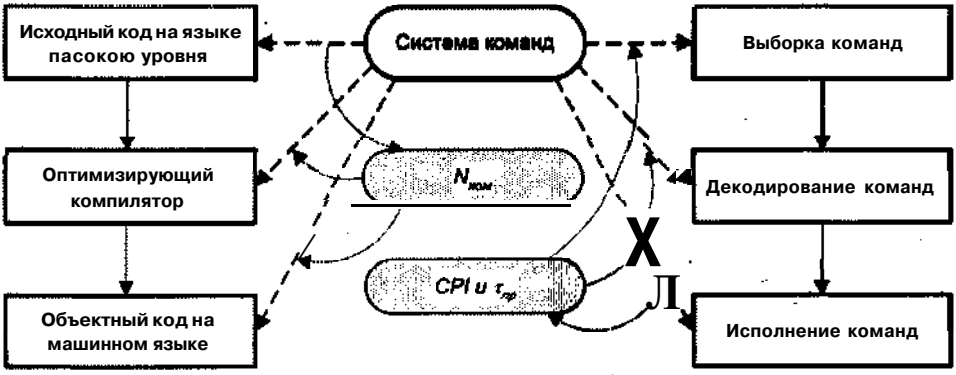


Рис. 2.2. Взаимосвязь между системой команд и факторами, определяющими эффективность вычислений

Общая характеристика архитектуры системы команд вычислительной машины складывается из ответов на следующие вопросы:

1. Какого вида данные будут представлены в вычислительной машине и в какой форме?
2. Где эти данные могут храниться помимо основной памяти?
3. Каким образом будет осуществляться доступ к данным?
4. Какие операции могут быть выполнены над данными?
5. Сколько операндов может присутствовать в команде?
6. Как будет определяться адрес очередной команды?
7. Каким образом будут закодированы команды?

Предметом данной главы является обзор наиболее распространенных архитектур системы команд, как в описательном плане, так и с позиций эффективности. В главе приводятся доступные статистические данные, позволяющие дополнить качественный анализ различных АСК количественными показателями. Большинство представленных статистических данных почерпнуто из общепризнанного источника - публикаций Д. Хеннеси и Д. Паттерсона. Данные были получены в результате реализации на вычислительной машине DEC VAX трех программных продуктов: компилятора с языка С GCC, текстового редактора TeX и системы автоматизированного проектирования Spice. Считается, что GCC и TeX показательны для программных приложений, где превалируют целочисленные вычисления и обработка текстов, а Spice может рассматриваться как типичный представитель вычислений с вещественными числами. С учетом того, что архитектура вычислительной машины VAX в известном смысле уже устарела, Хеннеси и Паттерсоном, а также приверженцами их методики были проведены дополнительные исследования, где программы GCC, Spice и TeX выполнялись на более современной ВМ, в частности MIPS R2000. Доступные данные для этого варианта также приводятся.

Классификация архитектур системы команд

В истории развития вычислительной техники как в зеркале отражаются изменения, происходившие во взглядах разработчиков на перспективность той или иной архитектуры системы команд. Сложившуюся на настоящий момент ситуацию в области АСК иллюстрирует рис. 2.3.

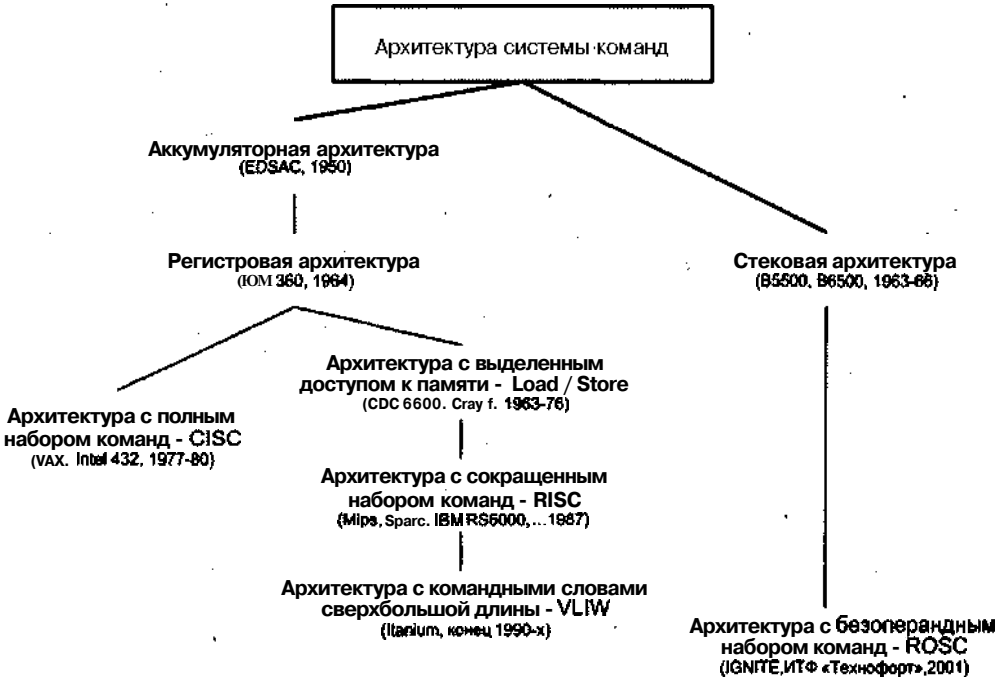


Рис. 2.3. Хронология развития архитектур системы команд

Среди мотивов, чаще всего предопределяющих переход к новому типу АСК, остановимся на двух наиболее существенных. Первый - это состав операций, выполняемых вычислительной машиной, и их сложность. Второй - место хранения операндов, что влияет на количество и длину адресов, указываемых в адресной части команд обработки данных. Именно эти моменты взяты в качестве критериев излагаемых ниже вариантов классификации архитектур системы команд.

Классификация по составу и сложности команд

Современная технология программирования ориентирована на языки высокого уровня (ЯВУ), главная цель которых — облегчить процесс программирования. Переход к ЯВУ, однако, породил серьезную проблему: сложные операторы, характерные для ЯВУ, существенно отличаются от простых машинных операций, реализуемых в большинстве вычислительных машин. Проблема получила название *семантического разрыва,* а ее следствием становится недостаточно эффективное

выполнение программ на ВМ. Пытаясь преодолеть семантический разрыв, разработчики вычислительных машин в настоящее время выбирают один из трех подходов и, соответственно, один из трех типов АС К:

- архитектуру с полным набором команд: CISC (Complex Instruction Set Computer);
- архитектуру с сокращенным набором команд: RISC (Reduced Instruction Set Computer);
- архитектуру с командными словами сверхбольшой длины:VLIW (Very Long Instruction Word).

В вычислительных машинах типа CISC проблема семантического разрыва решается за счет расширения системы команд, дополнения ее сложными командами, семантически аналогичными операторам ЯВУ. Основоположником CISC-архитектуры считается компания IBM, которая начала применять данный подход с семейства машин IBM 360 и продолжает его в своих мощных современных универсальных ВМ, таких как IBM ES/9000. Аналогичный подход характерен и для компании Intel в ее микропроцессорах серии 8086 и Pentium. Для CISC-архитектуры типичны:

- наличие в процессоре сравнительно небольшого числа регистров общего назначения;
- большое количество машинных команд, некоторые из них аппаратно реализуют сложные операторы ЯВУ;
- разнообразие способов адресации операндов;
- множество форматов команд различной разрядности;
- наличие команд, где обработка совмещается С обращением к памяти.

К типу CISC можно отнести практически все ВМ, выпускавшиеся до середины 1980-х годов, и значительную часть производящихся в настоящее время. Рассмотренный способ решения проблемы семантического разрыва вместе с тем ведет к усложнению аппаратуры ВМ, главным образом устройства управления, что, в свою очередь, негативно сказывается на производительности ВМ в целом. Это заставило более внимательно проанализировать программы, получаемые после компиляции с ЯВУ. Был предпринят комплекс исследований [128, 158, 177, 209], в результате которых обнаружилось, что доля дополнительных команд, эквивалентных операторам ЯВУ, в общем объеме программ не превышает 10-20%, а для некоторых наиболее сложных команд даже 0,2%. В то же время объем аппаратных средств, требуемых для реализации дополнительных команд, возрастает весьма существенно. Так, емкость микропрограммной памяти при поддержании сложных команд может увеличиваться на 60%.

Детальный анализ результатов упомянутых исследований привел к серьезному пересмотру традиционных решений, следствием чего стало появление *RISC-архитектуры*. Термин RISC впервые был использован Д. Паттерсоном и Д. Дитцелем в 1980 году [177]. Идея заключается в ограничении списка команд ВМ наиболее часто используемыми простейшими командами, оперирующими данными, размещенными только в регистрах процессорах. Обращение к памяти допускается лишь с

помощью специальных команд чтения и записи. Резко уменьшено количество форматов команд и способов указания адресов операндов. Сокращение числа форматов команд и их простота, использование ограниченного количества способов адресации, отделение операций обработки данных от операций обращения к памяти позволяет существенно упростить аппаратные средства ВМ и повысить их быстродействие. RISC-архитектура разрабатывалась таким образом, чтобы уменьшить $_{выч}$ за счет сокращения *CPI* и $\tau_{пр}$. Как следствие, реализация сложных команд за счет последовательности из простых, но быстрых RISC-команд оказывается не менее эффективной, чем аппаратный вариант сложных команд в CISC-архитектуре.

Элементы RISC-архитектуры впервые появились в вычислительных машинах CDC 6600 и суперЭВМ компании Cray Research. Достаточно успешно реализуется RISC-архитектура и в современных ВМ, например в процессорах Alpha фирмы DEC, серии PA фирмы Hewlett-Packard, семействе PowerPC и т. п.

Отметим, что в последних микропроцессорах фирмы Intel и AMD широко используются идеи, свойственные RISC-архитектуре, так что многие различия между CISC и RISC постепенно стираются.

Помимо CISC- и RISC-архитектур в общей классификации был упомянут еще один тип АСК - архитектура с командными словами сверхбольшой длины (VLIW). Концепция VLIW базируется на RISC-архитектуре, где несколько простых RISC-команд объединяются в одну сверхдлинную команду и выполняются параллельно. В плане АСК архитектура VLIW сравнительно мало отличается от RISC. Появился лишь дополнительный уровень параллелизма вычислений, в силу чего архитектуру VLIW логичнее адресовать не к вычислительным машинам, а к вычислительным системам.

Таблица 2.1. Сравнительная оценка CISC-, RISC- и VUW-архитектур

Характеристика	CISC	RISC	VLIW
Длина команды	Варьируется	Единая	Единая
Расположение полей в команде	Варьируется	Неизменное	Неизменное
Количество регистров	Несколько (часто специализированных)	Много регистров общего назначения	Много регистров общего назначения
Доступ к памяти	Может выполняться как часть команд различных типов	Выполняется только специальными командами	Выполняется только специальными командами -

Таблица 2.1 позволяет оценить наиболее существенные различия в архитектурах типа CISC, RISC и VLIW.

Классификация по месту хранения операндов

Количество команд и их сложность, безусловно, являются важнейшими факторами, однако не меньшую роль при выборе АСК играет ответ на вопрос о том, где могут храниться операнды и каким образом к ним осуществляется доступ. С этих позиций различают следующие виды архитектур системы команд:

- стековую;
- аккумуляторную;

, • регистровую;
• с выделенным доступом к памяти.

Выбор той или иной архитектуры влияет на принципиальные моменты: сколько адресов будет содержать адресная часть команд, какова будет длина этих адресов, насколько просто будет происходить доступ к операндам и какой, в конечном итоге, будет общая длина команд:

Стековая архитектура

Стеком называется память, по своей структурной организации отличная от основной памяти ВМ. Принципы построения стековой памяти детально рассматриваются позже, здесь же выделим только те аспекты, которые требуются для пояснения особенностей АСК на базе стека.

Стек образует множество логически взаимосвязанных ячеек (рис. 2.4), взаимодействующих по принципу «последним вошел, первым вышел» (LIFO, Last In First Out).

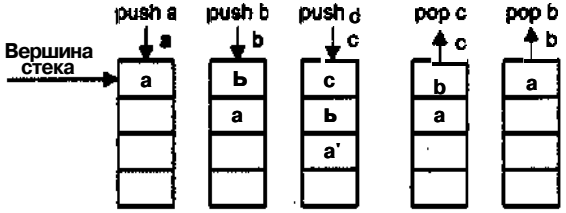


Рис. 2.4. Принцип действия стековой памяти

Верхнюю ячейку называют *вершиной стека*. Для работы со стеком предусмотрены две операции: *push* (проталкивание данных в стек) *upop* (выталкивание данных из стека). Запись возможна только в верхнюю ячейку стека, при этом вся хранящаяся в стеке информация предварительно проталкивается на одну позицию вниз. Чтение допустимо также только из вершины стека. Извлеченная информация удаляется из стека, а оставшееся его содержимое продвигается вверх. В вычислительных машинах, где реализована АСК на базе стека (их обычно называют стековыми), операнды перед обработкой помещаются в две верхних ячейки стековой памяти. Результат операции заносится в стек. Принцип действия стековой машины поясним на примере вычисления выражения $a = a + b + a \times c$.

При описании вычислений с использованием стека обычно используется иная форма записи математических выражений, известная как обратная польская запись (обратная польская нотация), которую предложил польский математик Я. Лукашевич. Особенность ее в том, что в выражении отсутствуют скобки, а знак операции располагается не между операндами, а следует за ними (постфиксная форма). Последовательность операций определяется их приоритетами (табл. 2.2).

Таблица 2.2. Приоритеты операций в обратной польской нотации

Операция	Символ операции	Приоритет
Открывающая скобка	(	0
Закрывающая скобка	)	1

продолжение

Таблица 2.2 *(продолжение)*

Операция	Символ операции	Приоритет
Сложение | вычитание	+ | -	2
Умножение | деление	* | /	3
Возведение в степень	**	4

При преобразовании традиционной записи выражения в постфиксную используется логическая структура, аналогичная стеку, которую, чтобы не путать ее со стеком вычислительной машины, назовем стеком последовательности операций (СПО). Формирование выходной строки с выражением в обратной польской нотации осуществляется в соответствии со следующим алгоритмом:

1. Исходная строка с выражением просматривается слева направо.
2. Операнды переписываются в выходную строку.
3. Знаки операций заносятся в СПО по следующим правилам:
   - если СПО пуст, то операция из входной строки переписывается в СПО;
   - операция выталкивает из СПО в выходную строку все операции с большим или равным приоритетом;
   - если очередной символ из исходной строки есть открывающая скобка, то он проталкивается в СПО;
   - закрывающая круглая скобка выталкивает все операции из СПО до ближайшей открывающей скобки, сами скобки в выходную строку не переписываются, а уничтожают друг друга.

Процесс получения обратной польской записи для правой части выражения $a = a + b + a * c$ представлен в табл. 2.3.

Таблица 2.3. Формирование обратной польской записи для выражения a = a + **b** + axc

Просматриваемый символ	1	2	3	4	5	6	7	8	9
Входная строка	a	+	b	+	a	X	c		
Состояние стека последовательности операций		+		+		X +		+	
Выходная строка	a		b	+	a		c	X	+

Таким образом, рассмотренное выше выражение в польской записи имеет вид: $a = ab+acx+$. Данная форма записи однозначно определяет порядок загрузки операндов и операций в стек (рис. 2.5).

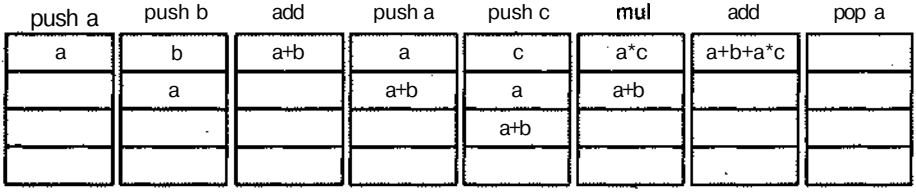


Рис. 2.5. Последовательность вычисления выражения a - ab+acx+ на вычислительной машине со стековой архитектурой

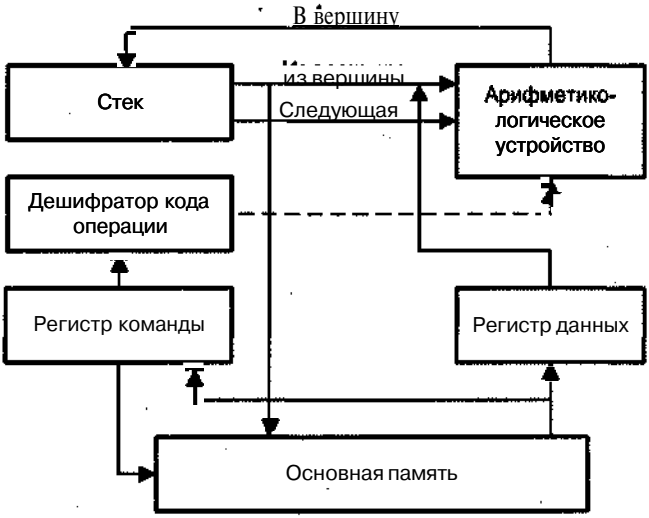


Рис. 2.6. Архитектура вычислительной машины на базе стека

Основные узлы и информационные тракты одного из возможных вариантов ВМ на основе стековой АСК показаны на рис. 2.6.

Информация может быть занесена в вершину стека из памяти или из АЛУ. Для Записи в стек содержимого ячейки памяти с адресом г выполняется команда *push x,* До которой информация считывается из ячейки памяти, заносится в регистр данных, а затем проталкивается в стек. Результат операции из АЛУ заносится в вершину стека автоматически.

Сохранение содержимого вершины стека в ячейке памяти с адресом *x* производится командой *pop x.* По этой команде содержимое верхней ячейки стека подается на шину, с которой и производится запись в ячейку *x,* после чего вся находящаяся в стеке информация проталкивается на одну позицию вверх.

Для выполнения арифметической или логической операции на вход АЛУ подается информация, считанная из двух верхних ячеек стека (при этом содержимое стека продвигается на две позиции вверх, то есть операнды из стека удаляются). Результат операции заталкивается в вершину стека. Возможен вариант, когда результат сразу же переписывается в память с помощью автоматически выполняемой операции *pop x.*

Верхние ячейки стековой памяти, где хранятся операнды и куда заносится результат операции, как правило, делаются более быстродействующими и размещаются в процессоре, в то время как остальная часть стека может располагаться в основной памяти и частично даже на магнитном диске.

К достоинствам АСК на базе стека следует отнести возможность сокращения адресной части команд, поскольку все операции производятся через вершину стека, то есть адреса операндов и результата в командах арифметической и логической обработки информации указывать не нужно. Код программы получается компактным. Достаточно просто реализуется декодирование команд.

С другой стороны, стековая АСК по определению не предполагает произвольного доступа к памяти, из-за чего компилятору трудно создать эффективный про-

граммный код, хотя создание самих компиляторов упрощается. Кроме того, стек становится «узким местом» ВМ в плане повышения производительности. В силу упомянутых причин, данный вид АСК долгое время считался неперспективным и встречался, главным образом, в вычислительных машинах 1960-х годов, например в ВМ фирмы Burroughs (B5500, B6500) или фирмы Hewlett-Packard (HP2116B, HP 3000/70).

Последние события в области вычислительной техники свидетельствуют о возрождении интереса к стековой архитектуре ВМ. Связано это с популярностью языка Java и расширением сферы применения языка Forth, семантике которых наиболее близка именно стековая архитектура. Среди современных ВМ со стековой АСК можно упомянуть машины JEM 1 и JEM 2 компании aJile Systems и Clip фирмы Imsys. Особо следует отметить стековую машину IGNITE компании Patriot Scientist, которую ее авторы считают представителем нового вида АСК — *'архитектурой с безоперандным набором команд.* Для обозначения таких ВМ они предлагают аббревиатуру ROSC (Removed Operand Set Computer). ROSC-архитектура заложена и в некоторые российские проекты, например разработки ИТФ «Технофорт». Строго говоря, по своей сути ROSC мало отличается от традиционной архитектуры на базе стека, и выделение ее в отдельный вид представляется не вполне обоснованным.

Аккумуляторная архитектура

Архитектура на базе аккумулятора исторически возникла одной из первых. В ней для хранения одного из операндов арифметической или логической операции в процессоре имеется выделенный регистр — *аккумулятор.* В этот же регистр заносится и результат операции. Поскольку адрес одного из операндов предопределен, в командах обработки достаточно явно указать местоположение только второго операнда. Изначально оба операнда хранятся в основной памяти, и до выполнения операции один из них нужно загрузить в аккумулятор. После выполнения команды обработки результат находится в аккумуляторе и, если он не является операндом для последующей команды, его требуется сохранить в ячейке памяти.

Типичная архитектура ВМ на базе аккумулятора показана на рис. 2.7.

Для загрузки в аккумулятор содержимого ячейки *x* предусмотрена команда загрузки *load x.* Поэтой команде информация считывается из ячейки памяти *x,* выход памяти подключается к входам аккумулятора и происходит занесение считанных данных в аккумулятор.

Запись содержимого аккумулятора в ячейку *x* осуществляется командой сохранения *store x,* при выполнении которой выходы аккумулятора подключаются к шине, после чего информация с шины записывается в память.

Для выполнения операции в АЛУ производится считывание одного из операндов из памяти в регистр данных. Второй операнд находится в аккумуляторе. Выходы регистра данных и аккумулятора подключаются к соответствующим входам АЛУ. По окончании предписанной операции результат с выхода АЛУ заносится в аккумулятор.

Достоинствами аккумуляторной АСК можно считать короткие команды и простоту декодирования команд. Однако наличие всего одного регистра порождает многократные обращения к основной памяти.

Рис. 2.7. Архитектура вычислительной машины на базе аккумулятора

АСК на базе аккумулятора была популярна в ранних ВМ, таких, например, как IBM 7090, DEC PDP-8, MOS 6502.

Регистровая архитектура

В машинах данного типа процессор включает в себя массив регистров (регистровый файл), известных как регистры общего назначения (РОН). Эти регистры, в каком-то смысле, можно рассматривать как явно управляемый кэш для хранения недавно использовавшихся данных.

Размер регистров обычно фиксирован и совпадает с размером машинного слова. К любому регистру можно обратиться, указав его номер. Количество РОН в архитектурах типа CISC обычно невелико (от 8 до 32), и для представления номера конкретного регистра необходимо не более пяти разрядов, благодаря чему в адресной части команд обработки допустимо одновременно указать номера двух, а зачастую и трех регистров (двух регистров операндов и регистра результата). RISC-архитектура предполагает использование существенно большего числа РОН (до нескольких сотен), однако типичная для таких ВМ длина команды (обычно 32 разряда) позволяет определить в команде до трех регистров.

Регистровая архитектура допускает расположение операндов в одной из двух запоминающих сред: основной памяти или регистрах. С учетом возможного размещения операндов в рамках регистровых АСК выделяют три подвида команд обработки:

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

В варианте «регистр-регистр» операнды могут находиться только в регистрах. В них же засылается и результат. Подтип «регистр-память» предполагает, что один

из операндов размещается в регистре, а второй в основной памяти. Результат обычно замещает один из операндов. В командах типа «память-память» оба операнда хранятся в основной памяти. Результат заносится в память. Каждому из вариантов свойственны свои достоинства и недостатки (табл. 2.4).

Таблица 2.4. Сравнительная оценка вариантов размещения операндов

Вариант	Достоинства	Недостатки
Регистр-регистр (0,3)	Простота реализации, фиксированная длина команд, простая модель формирования объектного кода при компиляции программ, возможность выполнения всех команд за одинаковое количество тактов	Большая длина объектного кода, из-за фиксированной длины команд часть разрядов в коротких командах не используется
Регистр-память (1.2)	Данные могут быть доступны без загрузки в регистры процессора, простота кодирования команд, объектный код получается достаточно компактным	Потеря одного из операндов при записи результата, длинное поле адреса памяти в коде команды сокращает место под номер регистра, что ограничивает общее число РОН. CPI зависит от места размещения операнда
Память-память (3, 3)	Компактность объектного кода, малая потребность в регистрах для хранения промежуточных данных	Разнообразие форматов команд и времени их исполнения, низкое быстродействие из-за обращения к памяти

В выражениях вида *(m, n),* приведенных в первом столбце таблицы, *m* означает количество операндов, хранящихся в основной памяти, а и - общее число операндов в команде арифметической или логической обработки.

Вариант «регистр-регистр» является основным в вычислительных машинах типа RISC. Команды типа «регистр-память» характерны для CISC-машин. Наконец, вариант «память-память» считается неэффективным, хотя и остается в наиболее сложных моделях машин класса CISC.

Возможную структуру и информационные тракты вычислительной машины с регистровой архитектурой системы команд иллюстрирует рис. 2.8.

Операции загрузки регистров из памяти и сохранения содержимого регистров в памяти идентичны таким же операциям с аккумулятором. Отличие состоит в этапе выбора нужного регистра, обеспечиваемого соответствующими селекторами.

Выполнение операции в АЛУ включает в себя:

- выбор регистра первого операнда;
- определение расположения второго операнда (память или регистр);
- подачу на вход АЛУ операндов и выполнение операции;
- выбор регистрарезультата и занесение в него результата операции из АЛУ.

Обратим внимание на то, что между АЛУ и регистровым файлом должны быть по крайней мере три шины.

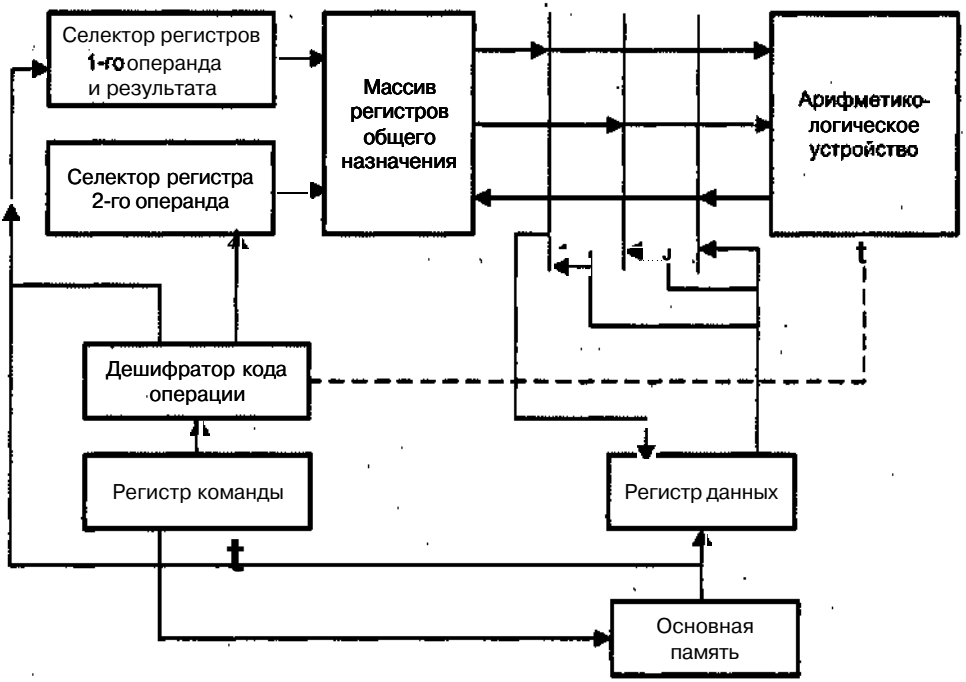


Рис. 2.8. Архитектура вычислительной машины на базе регистров общего назначения

К достоинствам регистровых АСК следует отнести: компактность получаемого кода, высокую скорость вычислений за счет замены обращений к основной памяти на обращения к быстрым регистрам. С другой стороны, данная архитектура требует более длинных инструкций по сравнению с аккумуляторной архитектурой.

Примерами машин на базе РОН могут служить CDC 6600, IBM 360/370, PDP-11, все современные персональные компьютеры. Правомочно утверждать, что в наши дни этот вид архитектуры системы команд является преобладающим.

Архитектура с выделенным доступом к памяти

В архитектуре с выделенным доступом к памяти обращение к основной памяти возможно только с помощью двух специальных команд: *load и store*. В английской транскрипции данную архитектуру называют Load/Store architecture. Команда *load* (загрузка) обеспечивает считывание значения из основной памяти и занесение его в регистр процессора (в команде обычно указывается адрес ячейки памяти и номер регистра). Пересылка информации в противоположном направлении производится командой *store* (сохранение). Операнды во всех командах обработки информации могут находиться только в регистрах процессора (чаще всего в регистрах общего назначения). Результат операции также заносится в регистр. В архитектуре отсутствуют команды обработки, допускающие прямое обращение к основной памяти. Допускается наличие в АСК ограниченного числа команд, где операнд является частью кода команды.

Состав и информационные тракты ВМ с выделенным доступом к памяти показаны на рис. 2.9. Две из трех шин, расположенных между массивом РОН и АЛУ,

обеспечивают передачу в арифметико-логическое устройство операндов, хранящихся в двух регистрах общего назначения. Третья служит для занесения результата в выделенный для этого регистр. Эти же шины позволяют загрузить в регистры содержимое ячеек основной памяти и сохранить в ОП информацию, находящуюся в РОН.

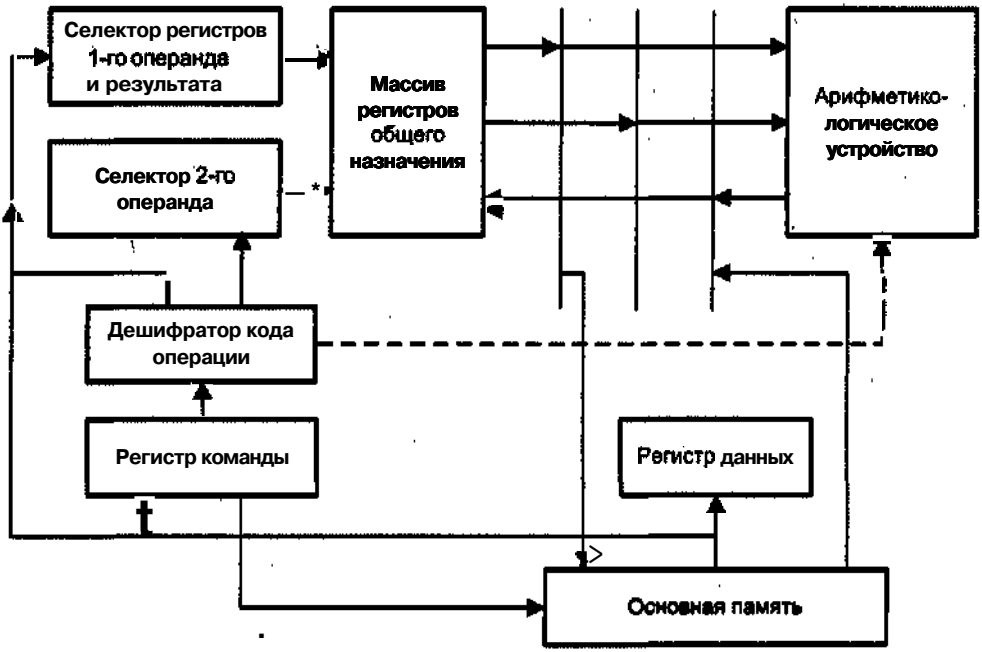


Рис. 2.9. Архитектура вычислительной машины с выделенным доступом к памяти

АСК с выделенным доступом к памяти характерна для всех вычислительных машин с RISC-архитектурой. Команды в таких ВМ, как правило, имеют длину 32 бита и трехадресный формат. В качестве примеров вычислительных машин с выделенным доступом к памяти можно отметить HP PA-RISC, IBM RS/6000, Sun SPARC, MIPS R4000, DEC Alpha и т. д. К достоинствам АСК следует отнести простоту декодирования и исполнения команды.

Типы и форматы операндов

Машинные команды оперируют данными, которые в этом случае принято называть *операндами*. К наиболее общим (базовым) типам операндов можно отнести: адреса, числа, символы и логические данные. Помимо них ВМ обеспечивает обработку и более сложных информационных единиц: графических изображений, аудио-, видео- и анимационной информации. Такая информация является производной от базовых типов данных и хранится в виде файлов на внешних запоминающих устройствах. Для каждого типа данных в ВМ предусмотрены определенные форматы.

Числовая информация

Среди цифровых данных можно выделить две группы:

- целые типы, используемые для представления целых чисел;
- вещественные типы для представления рациональных чисел.

В рамках первой группы имеется несколько форматов представления численной информации, зависящих от ее характера. Для представления вещественных чисел используется форма с плавающей запятой.

Числа в форме с фиксированной запятой

Представление числа X в *форме с фиксированной запятой* (ФЗ), которую иногда называют также *естественной формой,* включает в себя знак числа и его модуль в q-ичном коде. Здесь q - *основание системы счисления* или *база*. Для современных ВМ характерна двоичная система *(q = 2),* но иногда используются также восьмеричная $(q = 8)$ или шестнадцатеричная $(q = 16)$ системы счисления. Запятую в записи числа называют соответственно двоичной, восьмеричной или шестнадцатеричной. Знак положительного числа кодируется двоичной цифрой 0, а знак отрицательного числа - цифрой 1.

Числам с ФЗ соответствует запись вида $X = \pm a_{n-1}...a_1a_0a_{-1}a_{-2}...a_{-r}$. Отрицательные числа обычно представляются в дополнительном коде. Разряд кода числа, в котором размещается знак, называется *знаковым разрядом кода*. Разряды, где располагаются значащие цифры числа, называются *цифровыми разрядами кода*. Знаковый разряд размещается левее старшего цифрового разряда. Положение запятой одинаково для всех чисел и в процессе решения задач не меняется. Хотя запятая и фиксируется, в коде числа она никак не выделяется, а только подразумевается. В общем случае разрядная сетка ВМ для размещения чисел в форме с ФЗ имеет вид, представленный на рис. 2.10, где n разрядов используются для записи целой части числа и r разрядов — для дробной части.

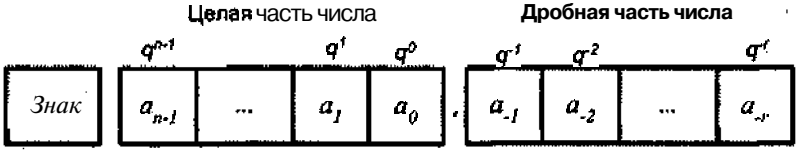


Рис. 2.10. Формат представления чисел с фиксированной запятой

При заданных значениях n и r диапазон изменения модулей чисел, коды которых могут быть представлены в данной разрядной сетке, определяется соотношением

$$Q^{-r} = |X| = q^n - q^{-r}.$$

Если число является смешанным (содержит целую и дробную части), оно обрабатываются как целое, хотя и не является таковым (в этом случае применяют термин *масштабируемоецелое)*. Обработка смешанных чисел в ВМ встречается крайне редко. Как правило, используются ВМ с дробной *(n = 0)* либо целочисленной *(r = 0)* арифметикой.

При фиксации запятой перед старшим цифровым разрядом (рис. 2.11) могут быть представлены только правильные дроби. Для ненулевых чисел возможны два варианта представления (нулевому значению соответствуют нули во всех разрядах): знаковое и беззнаковое. Фиксация запятой перед старшим разрядом встречалась в ряде машин второго поколения, но в настоящее время практически отжила свое.

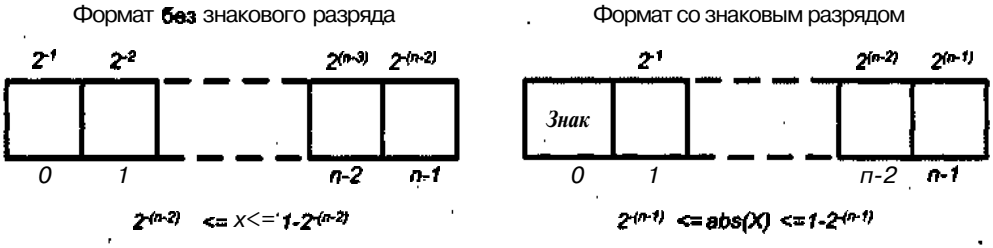


Рис. 2.11. Представление дробных чисел в формате ФЗ

При фиксации запятой после младшего разряда представимы лишь целые числа. Это наиболее распространенный способ, поэтому в дальнейшем понятие ФЗ будет связываться исключительно с целыми числами, а операции с числами в форме ФЗ будут характеризоваться как целочисленные. Здесь также возможны числа со знаком и без знака (рис. 2.12):

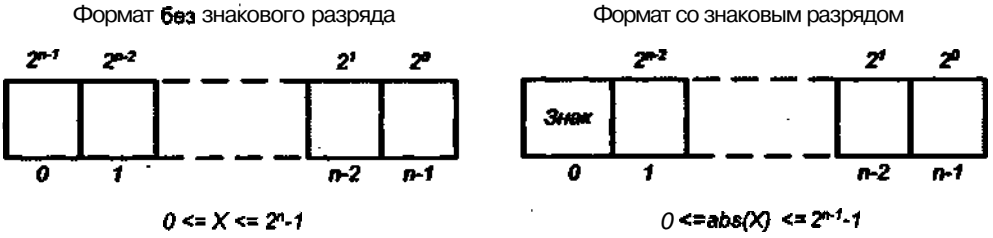


Рис. 2.12. Представление целых чисел в формате ФЗ

На рис. 2.13 приведены целочисленные форматы с фиксированной запятой, принятые в микропроцессорах фирмы Intel.

Целые числа применяются также для работы с адресами. На рис. 2.13 это 32-разрядный формат ближнего и 48-разрядный формат дальнего указателей.

Представление чисел в формате ФЗ упрощает аппаратурную реализацию ВМ и сокращает время выполнения машинных операций, однако при решении задач необходимо постоянно следить за тем, чтобы все исходные данные, промежуточные и окончательные результаты не выходили за допустимый диапазон формата, иначе возможно переполнение разрядной сетки и результат вычислений будет неверным.

Упакованные целые числа

В АСК современных микропроцессоров имеются команды, оперирующие целыми числами, представленными в упакованном виде. Связано это с обработкой муль-

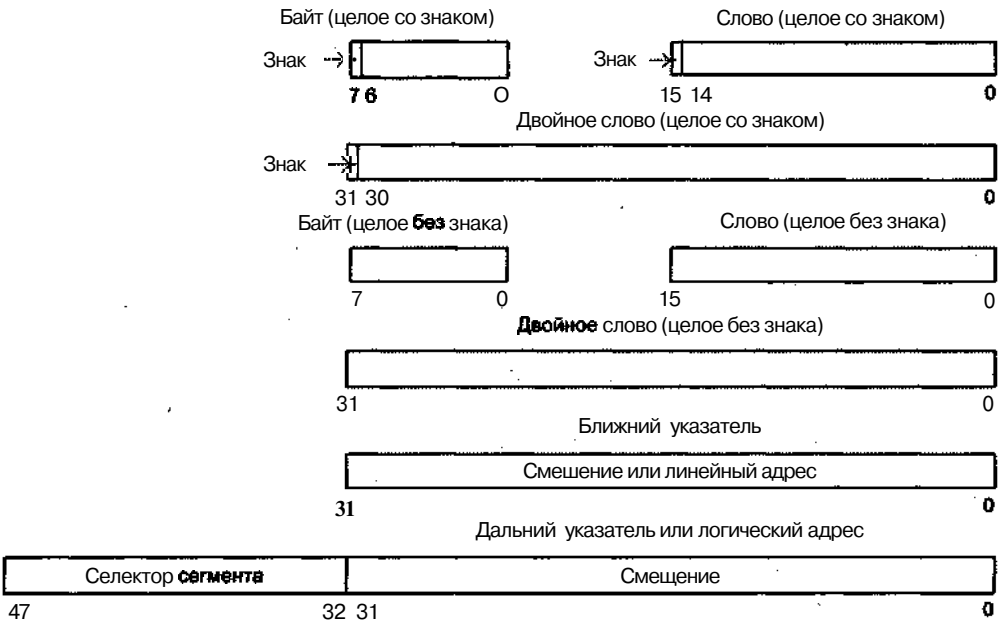


Рис. 2.13. Целочисленные форматы микропроцессоров фирмы Intel

тимедийной информации. Формат предполагает упаковку в пределах достаточно длинного слова (обычно 64-разрядного) нескольких небольших целых чисел, а соответствующие команды обрабатывают все эти числа параллельно. Если каждое из чисел состоит из четырех двоичных разрядов, то в 64-разрядное слово можно поместить до 16 таких чисел. Неиспользованные разряды заполняются нулями.

В микропроцессорах фирмы Intel, начиная с Pentium MMX, присутствуют специальные команды для обработки мультимедийной информации (MMX-команды), оперирующие целыми числами, упакованными в квадрослова (64-разрядные слова). Предусмотрены три формата (рис. 2.14): упакованные байты (восемь 8-разрядных чисел); упакованные слова (четыре 16-разрядных числа) и упакованные двойные слова (два 32-разрядных числа).

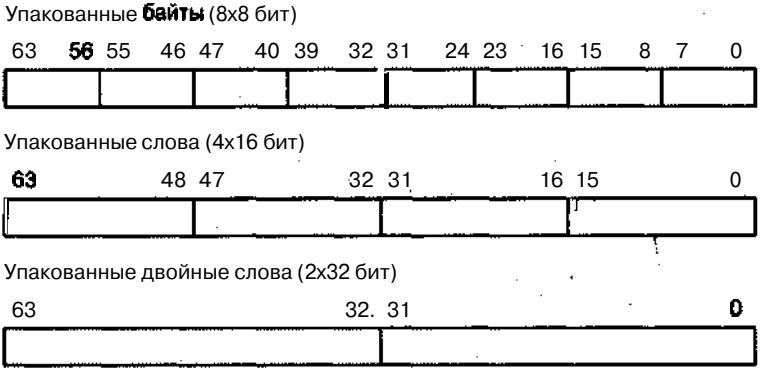


Рис. 2.14. Форматы упакованных целых чисел в технологиях MMX и 3DNow!

Байты в формате упакованных байтов нумеруются от 0 до 7, причем байт 0 располагается в младших разрядах квадрослова. Аналогичная система нумерации и размещения упакованных чисел применяется для упакованных слов (номера 0-3) и упакованных двойных слов (номера 0-1).

Идентичные форматы упакованных данных применяются также в другой технологии обработки мультимедийной информации, предложенной фирмой AMD. Эта технология носит название 3DNow!, а реализована в микропроцессорах данной фирмы.

Десятичные числа

В ряде задач, главным образом, учетно-статистического'характера, приходится иметь дело с хранением, обработкой и пересылкой десятичной информации. Особенность таких задач состоит в том, что обрабатываемые числа могут состоять из различного и весьма большого количества десятичных цифр. Традиционные методы обработки с переводом исходных данных в двоичную систему счисления и обратным преобразованием результата зачастую сопряжены с существенными накладными расходами. По этой причине в ВМ применяются иные специальные формы представления десятичных данных. В их основу положен принцип кодирования каждой десятичной цифры эквивалентным двоичным числом из четырех битов (тетрадой), то есть так называемым двоично-десятичным кодом (BCD - Binary Coded Decimal).

Байт	Байт		Байт	Байт
Зона | Цифра	Зона | Цифра	...	Зона | Цифра	Знак | Цифра

а

Байт	Байт		Байт	Байт
Цифра | Цифра	Цифра | Цифра	...	Цифра | Цифра	Цифра | Знак

б

Рис. **2.15.** Форматы десятичных чисел: а - зонный; *б* - уплотненный

Используются два формата представления десятичных чисел (все числа рассматриваются как целые): *зонный (распакованный)* и *уплотненный (упакованный)*. В обоих форматах каждая десятичная цифра представляется двоичной тетрадой, то есть заменяется двоично-десятичным кодом. Из оставшихся задействованных шести четырехразрядных двоичных комбинаций ($2^4 = 16$) две служат для кодирования знаков «+» и «-». Например, в ВМ семейства IBM 360/370/390 для знака «плюс» выбран код $1100_2 = C_{16}$, а для знака «минус» — код $1101_2 = D_{16}$.

Зонный формат (рис. 2.15, *а)* применяется в операциях ввода/вывода. В нем под каждую цифру выделяется один байт, где младшие четыре разряда отводятся под код цифры, а в старшую тетраду (поле зоны) записывается специальный код «зона», не совпадающий с кодами цифр и знаков. В IBM 360/370/390 это код $1111_2 = F_{16}$. Исключение составляет байт, содержащий младшую цифру десятичного числа, где в поле зоны хранится знак числа. На рис. 2.16 показана запись числа -7396 в зонном формате. В некоторых ВМ принят вариант зонного формата, где поле зоны заполняется нулями.

Байт		Байт		Байт		Байт	
Зона	7	Зона	3	Зона	9	Минус	6
1111	0111	1111	0011	1111	1001	1101	0110

Рис, 2.16. Представление числа -7396 в зонном формате

При выполнении операций сложения и вычитания над десятичными числами обычно используется упакованный формат и в нем же получается результат (умножение и деление возможно только в зонном формате). В упакованном формате (рис. 2.15, б) каждый байт содержит коды двух десятичных цифр. Правая тетрада последнего байта предназначается для записи знака числа. Десятичное число должно занимать целое количество байтов. Если это условие не выполняется, то четыре старших двоичных разряда левого байта заполняется нулями. Так, представление числа -7396 в упакованном формате имеет вид, приведенный на рис. 2.17.

Байт		Байт		Байт	
0	7	3	в	в	Минус
0000	0111	0011	1001	0110	1101

Рис. 2.17. Представление числа -7396 в упакованном формате

Размещение знака в младшем байте, как в зонном, так и в упакованном представлениях, позволяет задавать десятичные числа произвольной длины и передавать их в виде цепочки байтов. В этом случае знак указывает, что байт, в котором он содержится, является последним байтом данного числа, а следующий байт последовательности — это старший байт очередного числа.

Числа в форме с плавающей запятой

От недостатков ФЗ в значительной степени свободна форма представления чисел с *плавающей запятой* (ПЗ), известная также под названиями *нормальной* или *полулогарифмической* формы. В данном варианте каждое число разбивается на две группы цифр. Первая группа цифр называется *мантиссой,* вторая - *порядком.* Число представляется в виде произведения $X = \pm mq^{\pm p}$, где m — мантисса числа X, p — порядок числа, q — основание системы счисления.

Для представления числа в форме с ПЗ требуется задать знаки мантиссы и порядка, их модули в q-ичном коде, а также основание системы счисления (рис. 2.18). Нормальная форма неоднозначна, так как взаимное изменение m и p приводит к «плаванию» запятой, чем и обусловлено название этой формы.

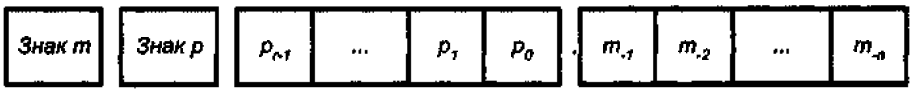


Рис. 2.18. Форма представления чисел с плавающей запятой

Диапазон и точность представления чисел с ПТ зависят от числа разрядов, отводимых под порядок и мантиссу. На рис. 2.19 показаны диапазоны разрядностей порядка и мантиссы, характерные для известных ВМ.

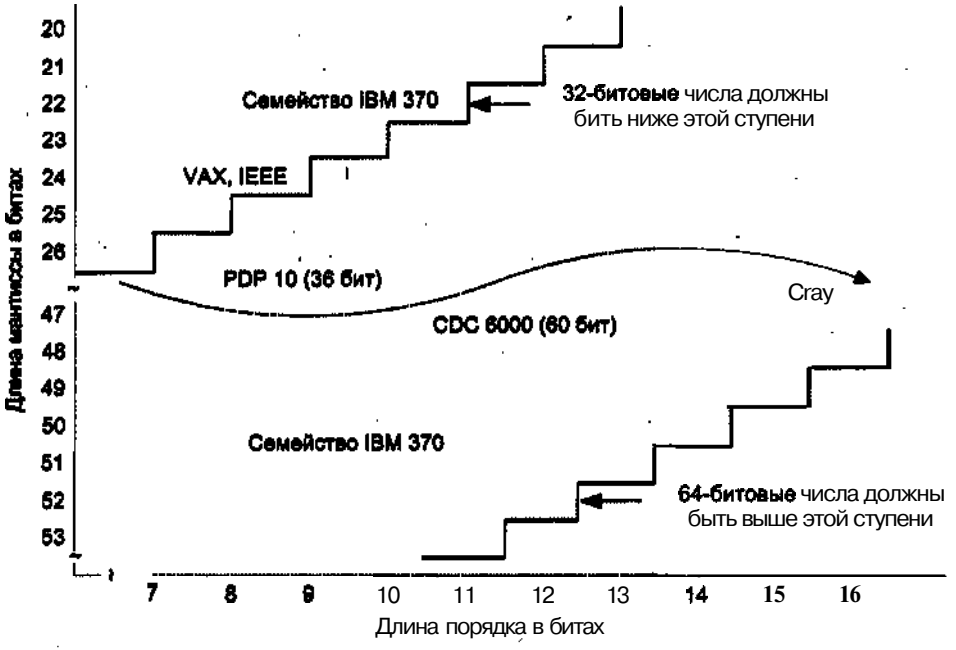


Рис. 2.19. Типовые разрядности полей порядка и мантиссы

Помимо разрядности порядка и мантиссы диапазон представления чисел зависит и от основания используемой системы счисления, которое может быть отличным от 2. Например, в универсальных ВМ (мэйнфреймах) фирмы IBM используется база 16. Это позволяет при одинаковом количестве битов, отведенных под порядок, представлять числа в большем диапазоне. Так, если поле порядка равно 7 битам, максимальное значение, на которое умножается мантисса, равно 2^{128} (при $q = 2$) или 16^{128} (при $q = 16$), а диапазоны представления чисел соответственно составят $10^{-19} < |X| < 10^{+19}$ и $10^{-76} < |X| < 10^{+76}$. Известны также случаи использования базы 8, например, в ВМ B-5500 фирмы Burroughs.

В большинстве вычислительных машин для упрощения операций над порядками последние приводят к целым положительным числам, применяя так называемый *смещенный порядок*. Для этого к истинному порядку добавляется целое положительное число — смещение (рис. 2.20). Например, в системе со смещением 128 порядок -3 представляется как 125 (-3 + 128). Обычно смещение выбирается равным половине представимого диапазона порядков. Отметим, что смещенный порядок занимает все биты поля порядка, в том числе и тот, который ранее использовался для записи знака порядка.

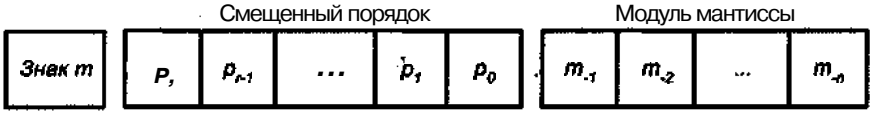


Рис. 2.20. Формат числа с ПЗ со смещенным порядком

Мантисса в числах с ПЗ обычно представляется в *нормализованной форме*. Это означает, что на мантиссу налагаются такие условия, чтобы она по модулю была меньше единицы ($|q| < 1$), а первая цифра после точки отличалась от нуля. Полученная таким образом мантисса называется *нормализованной*. Для применяемых в ВМ систем счисления можно записать:

- двоичная: $X = q2^p$, $(1 > |q| - S)$;
- -восьмеричная: $X = q8^p$, $(1 > |q| = ?)$;
- -шестнадцатеричная: $X = q16^p$, $(1 > |q| = 1/16)$.

Если первые i цифр мантиссы равны нулю, для нормализации ее нужно сдвинуть относительно запятой на i разрядов влево с одновременным уменьшением порядка на i единиц. В результате такой операции число не изменяется.

База	До нормализации		После нормализации	
	Порядок	Мантисса	Порядок	Мантисса
2	100	0,000110	001	0,110000
16	8	$0{,}001 \times 10^9$	6	$0{,}1 \times 1^{900}$

В примере для шестнадцатеричной системы после нормализации старшая цифра в двоичном представлении содержит впереди три нуля (0001). Это несколько уменьшает точность представления чисел по сравнению с двоичной системой при одинаковом числе двоичных разрядов, отведенных под мантиссу.

Если для записи числа с ПЗ используется база 2 ($q = 2$), то часто применяют еще один способ повышения точности представления мантиссы, называемый *приемом скрытой единицы*. Суть его в том, что в нормализованной мантиссе старшая цифра всегда равна единице (для представления нуля используется специальная кодовая комбинация), следовательно, эта цифра может не записываться, а подразумеваться. Запись мантиссы начинают с ее второй цифры, и это позволяет задействовать дополнительный значащий бит для более точного представления числа. Следует отметить, что значение порядка в данном случае не меняется. Скрытая единица перед выполнением арифметических операций восстанавливается, а при записи результата — удаляется. Таким образом, нормализованная мантисса 0,101000(1) при использовании способа «скрытой единицы» будет иметь вид 0,010001 (в скобках указана цифра, не поместившаяся в поле мантиссы при стандартной записи).

Для более существенного увеличения точности вычислений под число отводят несколько машинных слов, например два. Дополнительные биты, как правило, служат для увеличения разрядности мантиссы, однако в ряде случаев часть из них может отводиться и для расширения поля порядка. В процессе вычислений может получаться ненормализованное число. В таком случае ВМ, если это предписано командой, автоматически нормализует его.

Рассмотренные принципы представления чисел с ПЗ поясним на примере [200]. На рис. 2.21 представлен типичный 32-битовый формат числа с ПЗ. Старший (левый) бит содержит знак числа. Значение смещенного порядка хранится в разрядах

с 1-го по 8-й и может находиться в диапазоне от 0 до 255. Для получения фактического значения порядка из содержимого этого поля нужно вычесть фиксированное значение, равное 128. С таким смещением фактические значения порядка могут лежать в диапазоне от -128 до +127. В примере предполагается, что основание системы счисления равно 2. Третье поле слова содержит нормализованную мантиссу со скрытым разрядом (единицей). Благодаря такому приему 23-разрядное поле позволяет хранить 24-разрядную мантиссу в диапазоне of 0,5 до 1,0.

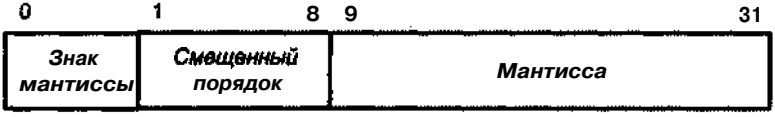


Рис. 2.21. Типичный 32-битовый формат числа с плавающей запятой

На рис. 2.22 приведены диапазоны чисел, которые могут быть записаны с помощью 32-разрядного слова.

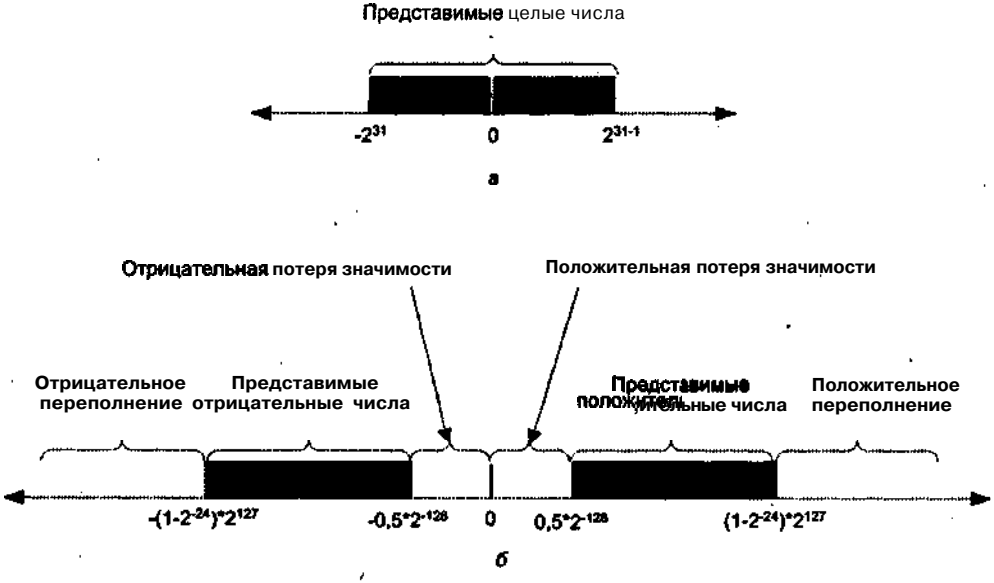


Рис. 2.22. Числа, **представимые** в 32-битовых форматах: а — целые числа с фиксированной запятой; *б* - числа с плавающей запятой

В варианте с ФЗ для целых чисел в дополнительном коде могут быть представлены все целые числа от -2^{31} до $2^{31} - 1$, то есть всего 2^{32} различных чисел (см. рис. 2.22, *а)*. Для случая ПЗ возможны следующие диапазоны чисел (см. рис. 2.22, *б*):

- отрицательные числа между $-(1 - 2^{-24}) \times 2^{127}$ и $-0{,}5 \times 2^{128}$;
- положительные числа между $0{,}5 \times 2^{-128}$ и $(1 - 2^{-24}) \times 2^{127}$.

В эту область не включены пять участков:

- отрицательные числа, меньшие чем $-(1 - 2^{24}) \times 2^{127}$ — *отрицательное переполнение;*

- отрицательные числа, большие чем $-0{,}5 \times 2^{-128}$ — *отрицательная потеря значимости;*
- положительные числа, меньшие чем $0{,}5 \times 2^{-28}$ — *положительная потеря значимости;*
- положительные числа, большие чем $(1 - 2^{-24}) \times 2^{127}$ — *положительное переполнение.*

Показанная запись числа с ПЗ не учитывает нулевого значения. Для этой цели используется специальная кодовая комбинация. Переполнения возникают, когда в результате арифметической операции получается значение большее, чем можно представить порядком 127 ($2^{120} \times 2^{100} = 2^{230}$). Потеря значимости — это когда результат представляет собой слишком маленькое дробное значение ($2^{-120} \times 2^{-100} = 2^{-230}$). Потеря значимости является менее серьезной проблемой, поскольку такой результат обычно рассматривают как нулевой.

Следует также отметить, что числа в форме с ПЗ, в отличие от чисел в форме с ФЗ, размещены на числовой оси неравномерно. Возможные значения в начале числовой оси расположены плотнее, а по мере движения вправо — все реже (рис. 2.23). Это означает, что многие вычисления приводят к результату, который не является точным, то есть представляет собой округление до ближайшего значения, представимого в данной форме записи.

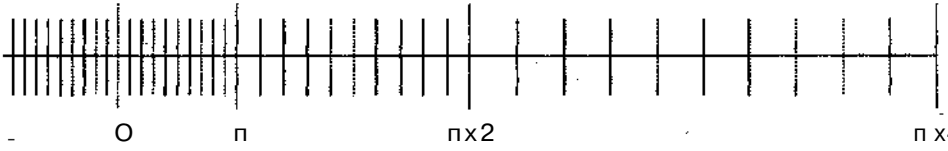


Рис. 2.23. Плотность чисел с плавающей запятой на числовой оси

Для формата, изображенного на рис. 2.21, имеет место противоречие между диапазоном и точностью. Если увеличить число битов, отведенных под порядок, расширяется диапазон представимых чисел. Однако, Поскольку может быть представлено только фиксированное число различных значений, уменьшается плотность и тем самым точность. Единственный путь увеличения как диапазона, так и точности — увеличение количества разрядов, поэтому в большинстве ВМ предлагается использовать числа в одинарном и двойном форматах. Например, число одинарного формата может занимать 32 бита, а двойного - 64 бита.

Числа с плавающей запятой в разных ВМ имеют несколько различных форматов. В табл. 2.5 приводятся основные параметры для нескольких систем представления чисел в форме с ПЗ. В настоящее время для всех ВМ рекомендован стандарт, разработанный общепризнанным международным центром стандартизации IEEE (Institute of Electrical and Electronics Engineers).

Таблица 2.5. Варианты форматов чисел с плавающей запятой[1]

Параметр	IBM 390	VAX	IEEE 794
Длина слова (бит)	О: 32; Д: 64	О: 32; Д: 64	О: 32; Д: 64
Порядок (бит)	7 бит	8 бит	О: 8; Д: 11

продолжение

[1] О — одинарный формат; Д — двойной формат.

Таблица 2.5 *(продолжение)*

Параметр	IBM 390	VAX	IEEE 754
Мантисса (F)	О: 6 цифр; Д: 14 цифр	О: (1) + 23 бита Д:(1) + 55 бит	0:<1) + 23 бита Д: (1) + 52 бита
Смещение порядка	64	128	К: 127; Д: 1023
База	16	2	2
Скрытая 1	Нет	Да	Да
Запятая	Слева от мантиссы	Слева от скрытой 1	Справа старшего бита мантиссы
Диапазон F	$(1,16) \le F < 1$	$0,5 \le F < 1$	$1 \le F < 2$
Представление F	Величина со знаком	Величина со знаком	Величина со знаком
Максимальное положительное число	$1663 \cong 1076$	$2126 \cong 1038$	21 024 $\cong$ 10 308 (Д)
Точность	О: $16^{-6} \cong 10^{-7}$ Д: $16^{-14} \cong 10^{-17}$	О: $2^{-24} \cong 10^{-7}$ Д: $2^{-564} \cong 10^{-17}$	О: $2^{-23} \cong 10^{-7}$ Д: $2^{-524} \cong 10^{-16}$

Стандарт IEEE 754

Рекомендуемый для всех ВМ формат представления чисел с плавающей запятой определен стандартом IEEE 754. Этот стандарт был разработан с целью облегчить перенос программ с одного процессора на другие и нашел широкое применение практически во всех процессорах и арифметических сопроцессорах.

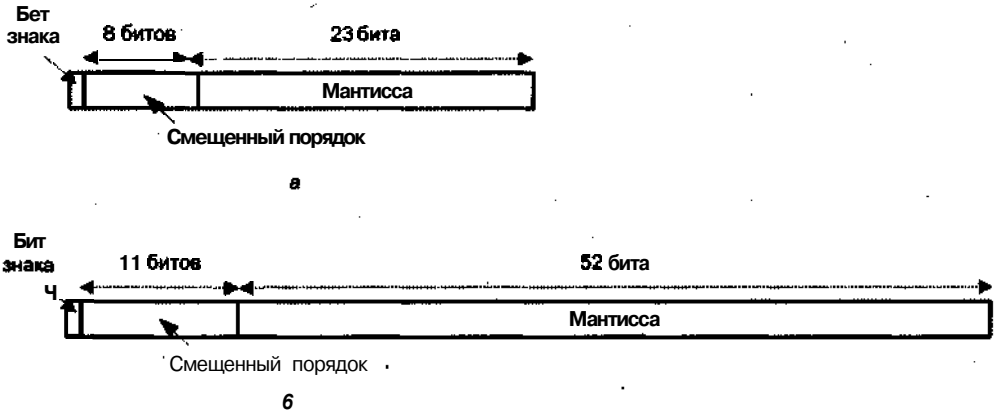


Рис. 2.24. Основные форматы IEEE754: *а* - одинарный; *б* - двойной

Стандарт определяет 32-битовый (одинарный) и 64-битовый (двойной) форматы (рис. 2.24) с 8- и 11-разрядным порядком соответственно. Основанием системы счисления является 2. В дополнение, стандарт предусматривает два расширенных формата, одинарный и двойной, фактический вид которых зависит от конкретной реализации. Расширенные форматы предусматривают дополнитель-

ные биты для порядка (увеличенный диапазон) и **мантиссы** (повышенная точность). Таблица 2.6 содержит описание основных характеристик всех четырех форматов.

Не все кодовые комбинации в форматах IEEE интерпретируются обычным путем — некоторые комбинации используются для представления специальных значений. Предельные значения порядка, содержащие все нули (0) и все единицы (255 — в одинарном формате и 2047 — в двойном формате), определяют специальные значения.

Таблица 2.6. Параметры форматов стандарта IEEE754

Параметр	Формат			
	одинарный	одинарный расширенный	двойной	двойной расширенный
Ширина слова, бит	32	>43	64	>79
Ширина порядка, бит..	8	>11	11	>15
Смещение порядка	127	Не определено	1023	Не определено
Максимальный порядок	127	> 1023	1023	> 16 383
Минимальный порядок	-126	<-1022	-1022	<-16382
Диапазон чисел	10^{-38}, 10^{+38}	Не определен	^{08}, 10308	Не определен
Длина мантиссы, бит	23	>31	52	>63
Количество порядков	254	Не определено	2046	Не определено
Количество мантисс	223	Не определено	252	Не определено
Количество значений	1,98 $x2^{31}$	Не определено	1,99 x 2^{63}	Не определено

Представлены следующие классы чисел:

- Порядки в диапазоне от 1 до 254 для одинарного формата и от 1 до 2036 — для двойного формата, используются для представления ненулевых нормализованных чисел. Порядки смещены так, что их диапазон составляет от -126 до +127 для одинарного формата и от -1022 до +1023 - для двойного формата. Нормализованное число требует, чтобы слева от двоичной запятой был единичный бит. Этот бит подразумевается, благодаря чему обеспечивается эффективная ширина мантиссы, равная 24 битам для одинарного и 53 битам — для двойного форматов.
- Нулевой порядок совместно с нулевой мантиссой представляют положительный или отрицательный 0, в зависимости от состояния бита знака мантиссы.
- Порядок, содержащий единицы во всех разрядах, совокупно с нулевой мантиссой представляют положительную или отрицательную бесконечность, в зависимости от состояния бита знака, что позволяет пользователю самому решить, считать ли это ошибкой или продолжать вычисления со значением, равным бесконечности.

- Нулевой порядок в сочетании с ненулевой мантиссой представляют ненормализованное число. В этом случае бит слева от двоичной точки равен 0 и фактический порядок равен -126 или -1022. Число является положительным или отрицательным в зависимости от значения знакового бита.
- Кодовая комбинация, в которой порядок содержит все единицы, а мантисса не равна 0, используется как признак «не числа» (NAN — Not a Number) и служит для предупреждения о различных исключительных ситуациях.

Упакованные числа с плавающей запятой

В последних версиях АСК, предусматривающих особые команды для обработки мультимедийной информации, помимо упакованных целых чисел используются и упакованные числа с плавающей запятой. Так, в уже упоминавшейся технологии 3DNow! фирмы AMD имеются команды, служащие для увеличения производительности систем при обработке трехмерных приложений, описываемых числами с ПЗ. Каждая такая команда работает с двумя операндами с плавающей запятой одинарной точности. Операнды упаковываются в 64-разрядные группы, как это показано на рис. 2.25.

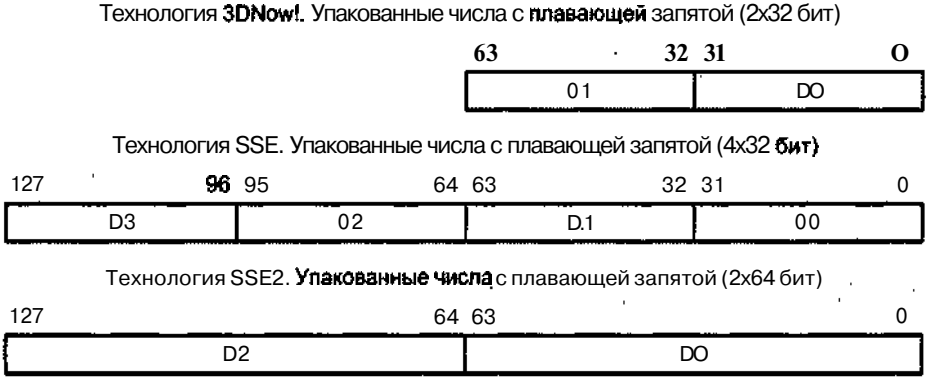


Рис. 2.25. Формат упакованных чисел с плавающей запятой в технологиях 3DNow!, SSE, SSE2

В микропроцессорах фирмы Intel, начиная с Pentium III, для аналогичных целей поддержаны команды, реализующие технологию SSE, также ориентированную на параллельную обработку упакованных чисел с ПЗ. Здесь числа объединяются в группы длиной 128 бит, и это позволяет упаковать в группу четыре 32-разрядных числа с ПЗ (числа с одинарной точностью). Позже, в технологии SSE2, которую можно считать дальнейшим развитием SSE, появился формат, где в группу из 128 бит упаковываются два 64-разрядных числа с ПЗ, то есть числа, представленные с двойной точностью.

Разрядность основных форматов числовых данных

Данные, представляющие в ВМ числовую информацию, могут иметь *фиксированную* или *переменную* длину. Операционные устройства вычислительных машин (целочисленные арифметико-логические устройства, блоки обработки чисел

с плавающей запятой, устройства десятичной арифметики и т. п.), как правило, рассчитаны на обработку кодов фиксированной длины. Общепринятые величины разрядности кодов чисел показаны на рис. 2.26.

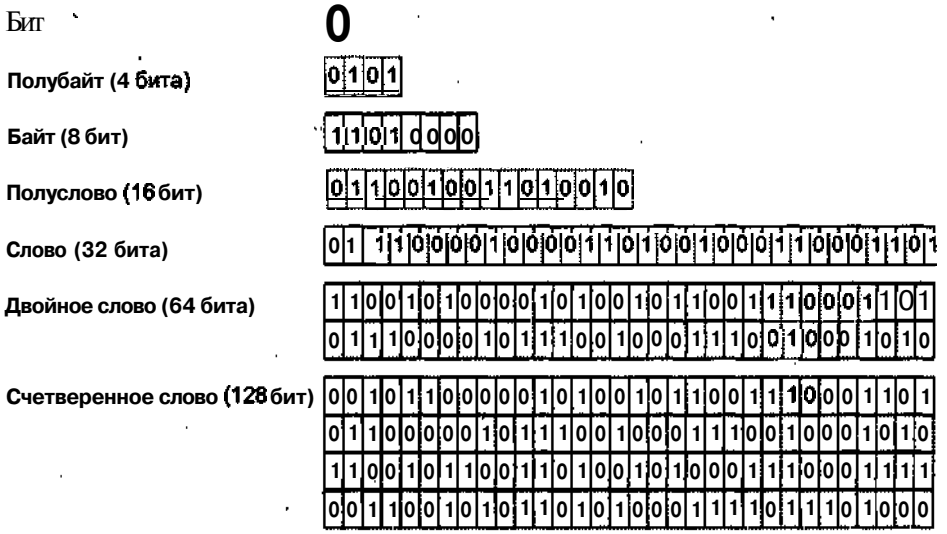


Рис. 2.26. Стандартная длина фиксированных форматов представления чисел

Наименьшей единицей данных в ВМ служит бит (BIT, Binary digiT - двоичная цифра). В большинстве случаев эта единица информации слишком мала. Однобитовые операционные устройства использовались в ВМ с последовательной обработкой информации, а в современных машинах с параллельной обработкой разрядов они практически не применяются. Побитовую работу с данными скорее можно встретить в многопроцессорных вычислительных системах, построенных из одноразрядных процессоров.

Следующая по величине единица состоит из четырех битов и называется полубайтом или тетрадой, или реже «ниблом» (nibble - огрызок). Она также редко имеет самостоятельное значение и заслуживает упоминания как единица представления отдельных десятичных цифр при их двоично-десятичной записи.

Реально наименьшей обрабатываемой единицей считается байт, состоящий из восьми битов. На практике эта единица информации также оказывается недостаточной, и значительно чаще применяются числа, представленные двумя (полуслово), четырьмя (слово), восемью (двойное слово) или шестнадцатью (счетверенное слово) байтами[1].

Разрядность целочисленного АЛУ обычно выбирается равной ширине адреса (для большинства современных ВМ это 32 разряда). Следовательно, наиболее выгодными в плане быстродействия являются такие целые числа, длина которых

[1] Согласно терминологии Intel. В другом варианте (DEC) естественный размер слова - 2 байта, двойного или длинного слова (longword) - 4 байта и упоминавшегося выше квадрослова (quadword) - 8 байт.

совпадает с разрядностью адреса. Использование более коротких чисел позволяет сэкономить на памяти, но выигрыша в производительности не дает.

Блоки операций с плавающей запятой обычно согласованы со стандартом IEEE 754 и рассчитаны на обработку чисел в формате двойной длины (64 бита). В большинстве ВМ реальная разрядность таких блоков даже больше (80 бит). Таким образом, наилучшим вариантом при проведении вычислений с плавающей запятой можно считать формат двойного слова. При выборе формата меньшей длины (32 разряда) вычисления все равно ведутся с большей точностью, после чего результат округляется. Таким образом, использование короткого формата чисел с плавающей запятой, как и в случае целых чисел с фиксированной запятой, помимо экономии памяти никаких иных преимуществ также не дает.

В работе [120] приводятся усредненные данные о частоте использования основных форматов чисел, полученные в ходе выполнения пакета тестовых программ SPEC92 на вычислительной машине DEC VAX (рис. 2.27).

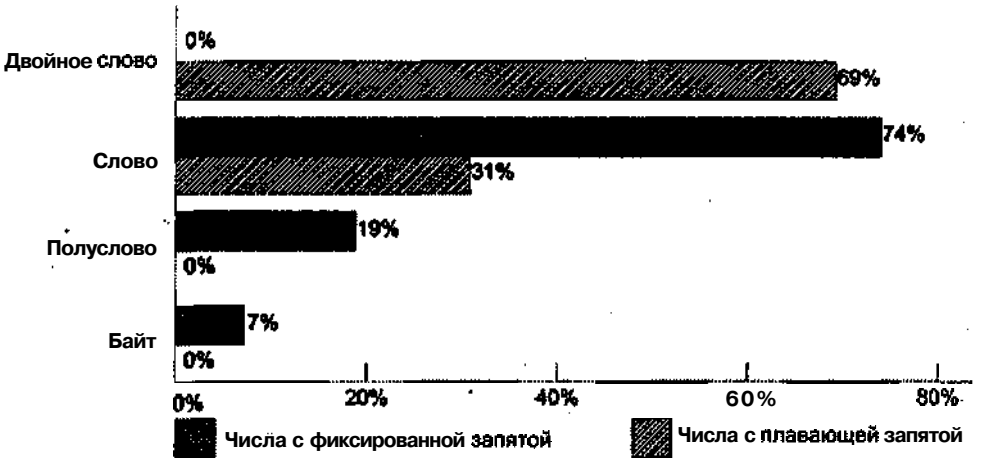


Рис. 2.27. Частота обращения к числовым данным в зависимости от их разрядности

В приложениях, оперирующих десятичными числами, где количество цифр в числе может варьироваться в широком диапазоне, что характерно для задач из области экономики, более удобными оказываются форматы переменной длины. В этом случае числа не переводятся в двоичную систему, а записываются в виде последовательности двоично-кодированных десятичных цифр. Длина подобной цепочки может быть произвольной, а для указания ее границы обычно используют символ-ограничитель, код которого не совпадает с кодами цифр. Длина цифровой последовательности может быть задана явно в виде количества цифр числа и храниться в первом байте записи числа, однако этот прием более характерен для указания длины строки символов.

Размещение числовых данных в памяти

В современных ВМ разрядность одной ячейки памяти, как правило, равна одному байту (8 бит). В то же время реальная длина кодов чисел составляет 2, 4, 8 или 16 байт. При хранении таких чисел в памяти последовательные байты числа раз-

мещают в нескольких ячейках с последовательными адресами, при этом для доступа к числу указывается только наименьший из адресов. При разработке архитектуры системы команд необходимо определить порядок размещения байтов в памяти, то есть какому из байтов (старшему или младшему) будет соответствовать этот наименьший адрес[1]. На рис. 2.28 показаны оба варианта размещения 32-разрядного числа в четырех последовательных ячейках памяти, начиная с адреса *x*.

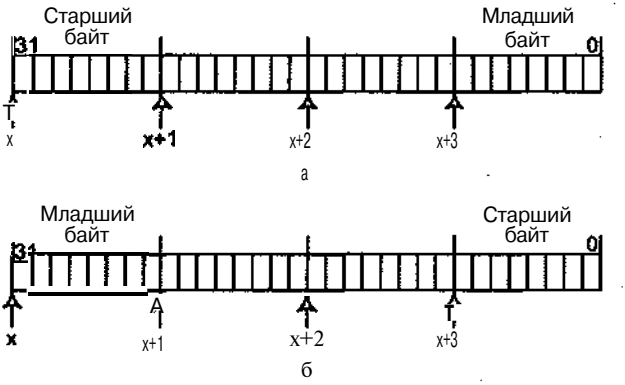


Рис. 2.28. Размещение в памяти 32-разрядного числа: а - начиная со старшего байта; *б* — начиная с младшего байта

В вычислительном плане оба способа записи равноценны. Так, фирмы DEC и Intel отдают предпочтения размещению в первой ячейке младшего байта, а IBM и Motorola ориентируются на противоположный вариант. Выбор обычно связан с некими иными соображениями разработчиков ВМ. В настоящее время в большинстве машин предусматривается использование обоих вариантов, причем выбор может быть произведен программным путем за счет соответствующей установки регистра конфигурации.

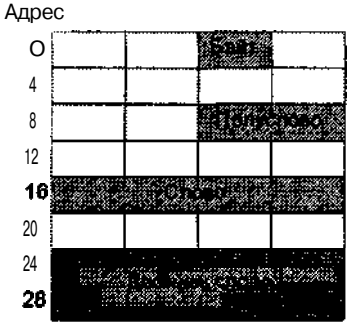


Рис. 2.29. Размещение чисел в памяти с выравниванием

[1] В англоязычной литературе систему записи числа, начиная со старшего байта, обозначают термином «big **endian**», а с младшего байта — термином «little **endian**». Оба названия происходят от названия племен (**«тупоконечники»** и «остроконечники»), упоминаемых в книге Джонатана Свифта «Путешествия Гулливера». Там описывается религиозная война между этими племенами, по причине разногласий в вопросе, с какого конца следует разбивать яйцо — тупого или острого.

Помимо порядка размещения байтов, существенным бывает и выбор адреса, с которого может начинаться запись числа. Связано это с физической реализацией полупроводниковых запоминающих устройств, где обычно предусматривается возможность считывания (записи) четырех байтов подряд. Причем данная операция выполняется быстрее, если адрес первого байта A отвечает условию $A \bmod 5 = 0$ ($S = 2$, 4, 8, 16). Числа, размещенные в памяти в соответствии с этим правилом, называются *выравненными* (рис. 2.29).

На рис. 2.30 показаны варианты размещения 32-разрядного слова без выравнивания. Их использование может приводить к снижению производительности.

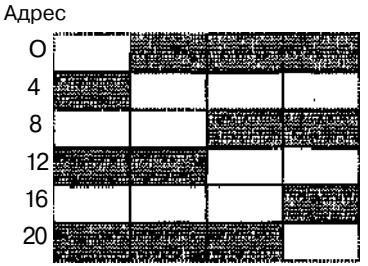


Рис. 2.30. Размещение 32-разрядного слова без соблюдения правила выравнивания

Большинство компиляторов генерируют код, в котором предусмотрено выравнивание чисел в памяти.

Символьная информация

В общем объеме вычислительных действий все большая доля приходится на обработку символьной информации, содержащей буквы, цифры, знаки препинания, математические и другие символы. Каждому символу ставится в соответствие определенная двоичная комбинация. Совокупность возможных символов и назначенных им двоичных кодов образует *таблицу кодировки*. В настоящее время применяется множество различных таблиц кодировки. Объединяет их весовой принцип, при котором веса кодов цифр возрастают по мере увеличения цифры, а веса символов увеличиваются в алфавитном порядке. Так вес буквы «Б» на единицу больше веса буквы «А». Это способствует упрощению обработки в ВМ.

До недавнего времени наиболее распространенными были кодовые таблицы, в которых символы кодируются с помощью восьмиразрядных двоичных комбинаций (байтов), позволяющих представить 256 различных символов:

- расширенный двоично-кодированный код EBCDIC (Extended Binary Coded Decimal Interchange Code);
- американский стандартный код для обмена информацией ASCII (American Standard Code for Information Interchange).

Код EBCDIC используется в качестве внутреннего кода в универсальных ВМ фирмы IBM. Он же известен под названием ДКОИ (двоичный код для обработки информации).

Стандартный код ASCII — 7-разрядный, восьмая позиция отводится для записи бита четности. Это обеспечивает представление 128 символов, включая все

латинские буквы, цифры, знаки основных математических операций и знаки пунктуации. Позже появилась европейская модификация ASCII, называемая Latin 1 (стандарт ISO 8859-1). В ней «полезно» используются все 8 разрядов. Дополнительные комбинации (коды 128-255) в новом варианте отводятся для представления специфических букв алфавитов западно-европейских языков, символов псевдографики, некоторых букв греческого алфавита, а также ряда математических и финансовых символов. Именно эта кодовая таблица считается мировым стандартом де-факто, который применяется с различными модификациями во всех странах. В зависимости от использования кодов 128-255 различают несколько вариантов стандарта ISO 8859 (табл. 2.7).

Таблица 2.7. Варианты стандарта ISO 8859

Стандарт	Характеристика
ISO 8859-1	Западно-европейские языки
ISO 8859-2	Языки стран центральной и восточной Европы
ISO 8859-3	Языки стран южной Европы, мальтийский и эсперанто
ISO 8859-4	Языки стран северной Европы
ISO 8859-5	Языки славянских стран с символами кириллицы
ISO 8859-6	Арабский язык
ISO 8859-7	Современный греческий язык
ISO 8859-8	Языки иврит и идиш
ISO 8859-9	Турецкий язык
ISO 8859-10	Языки стран северной Европы (лапландский, исландский)
ISO 8859-11	Тайский язык
ISO 8859-13	Языки балтийских стран
ISO 8859-14	Кельтский язык
ISO 8859-15	Комбинированная таблица для европейских языков
ISO 8859-16	Содержит специфические символы ряда языков: албанского, хорватского, английского, финского, французского, немецкого, венгерского, ирландского, итальянского, польского, румынского и словенского

В популярной в свое время операционной системе MS-DOS стандарт ISO 8859 реализован в форме *кодовых страниц* OEM (Original Equipment Manufacturer). Каждая OEM-страница имеет свой идентификатор (табл. 2.8).

Таблица 2.8. Наиболее распространенные кодовые страницы OEM

Идентификатор	Страны кодовой страницы
CP437	США, страны западной Европы и Латинской Америки
CP708	Арабские страны

продолжение

Таблица 2.8 ***(продолжение)***

Идентификатор	Страны кодовой страницы
CP737	Греция
CP775	Латвия, Литва, Эстония
CP852	Страны восточной Европы
CP853	Турция
CP855	Страны с кириллической письменностью
CP860	Португалия
CP862	Израиль
CP865	Дания, Норвегия
CP866	Россия
CP932	Япония
CP936	Китай

Хотя код ASGII достаточно удобен, он все же слишком тесен и не вмещает множества необходимых символов. По этой причине в 1993 году консорциумом компаний Apple Computer, Microsoft, Hewlett-Packard, DEC и IBM был разработан 16-битовый, стандарт ISO 10646, определяющий универсальный набор символов (UCS, Universal Character Set). Новый код, известный под названием Unicode, позволяет задать до 65 536 символов, то есть дает возможность одновременно представить символы всех основных «живых» и «мертвых» языков. Для букв русского языка выделены коды 1040-1093.

В «естественном» варианте кодировки Unicode, известном как UCS-2, каждый символ описывается двумя последовательными байтами *mun,* так что номеру символа соответствует численное значение $256 \times m + n$. Таким образом, кодовый номер представлен 16-разрядным двоичным числом. Наряду с UCS-2 в рамках Unicode существуют еще несколько вариантов кодировки Unicode (UTF, Unicode Transformation Formats), основные из которых UTF-8 и UTF-7.

В кодировке UTF-8 коды символов меньшие, чем 128, представляются одним байтом. Все остальные коды формируются по более сложным правилам. В зависимости от символа его код может занимать от двух до шести байтов, причем старший бит каждого байта всегда имеет единичное значение. Иными словами, значение байта лежит в диапазоне от 128 до 255. Ноль в старшем бите байта означает, что код занимает один байт и совпадает по кодировке с ASCII. Схема формирования кодов UTF- 8 показана в табл. 2.9.

Таблица 2.9. Структура кодов UTF-8

Число байтов	Двоичное представление	Число свободных битов
1	0xxxxxxx	7
2	110xxxxx 10xxxxxx	11(5 + 6)

Число байтов	Двоичное представление	Число свободных битов
3	110xxxx10xxxxxx 10xxxxxx	16(4 + 6x2)
4	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx	21(3 + 6x3)
5	11111Oxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx	26(2 + 6x4)
6	1111110x10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx	31(1+6x5)

В UTF-7 код символа также может занимать один или более байтов, однако в каждом из байтов значение не.превышает 127 (старший бит байта содержит ноль). Многие символы кодируются одним байтом, и их кодировка совпадает с ASCII, однако некоторые коды зарезервированы для использования в качестве преамбулы, характеризующей последующие байты многобайтового кода.

Стандарт Unicode обратно совместим с кодировкой ASCII, однако если в ASCII для представления схожих по виду символов (минус, тире, знак переноса) применялся общий код, в Unicode каждый из этих символов имеет уникальную кодировку. Впервые Unicode был использован в операционной системе Windows NT. Распределение кодов в Unicode иллюстрирует табл. 2.10.

Таблица 2.10. Блоки символов в стандарте Unicode

Коды	Символы
0-8191	Алфавиты - английский, европейские, фонетический, кириллица, армянский, иврит, арабский, эфиопский, бенгали, деванагари, гур, гуджарати, ория, телугу, тамильский, каннада, малайский, сингальский, грузинский, тибетский, тайский, лаосский, кхмерский, монгольский
8192-12287	Знаки пунктуации, математические операторы, технические символы, орнаменты и т. п.
12288-16383	Фонетические символы китайского, корейского и японского языков
16384-59391	Китайские, корейские, японские идеографы. Единый набор символов каллиграфии хань
59392-65024	Блок для частного использования
65025-65536	Блок обеспечения совместимости с программным обеспечением

Параллельно с развитием Unicode исследовательская группа ISO проводит работы над 32-битовой кодовой таблицей, однако ввиду широкой распространенности кодировки Unicode дальнейшие перспективы новой разработки представляются неопределенными.

Логические данные

Элементом логических данных является логическая (булева) переменная, которая может принимать лишь два значения: «истина» или «ложь». Кодирование логического значения принято осуществлять битом информации: единицей кодируют

истинное значение, нулем — ложное. Как правило, в ВМ оперируют наборами логических переменных длиной в машинное слово. Обрабатываются такие слова с помощью команд логических операций (И, ИЛИ, НЕ и т. д.), при этом все биты обрабатываются одинаково, но независимо друг от друга, то есть никаких переносов между разрядами не возникает.

Строки

Строки - это непрерывная последовательность битов, байтов, слов или двойных слов. *Битовая строка* может начинаться в любой позиции байта и содержать до 2^{32} бит. *Байтовая строка* может состоять из байтов, слов или двойных слов. Длина такой строки варьируется от нуля до 2^{32} - 1 байт (4 Гбайт). Приведенные цифры характерны для превалирующих в настоящее время 32-разрядных ВМ.

Если байты байтовой строки представляют собой коды символов, то говорят о *текстовой строке*. Поскольку длина текстовой строки может меняться в очень широких пределах, то для указания конца строки в последний байт заносится код-ограничитель - обычно это нули во всех разрядах байта. Иногда вместо ограничителя длину строки указывают числом, расположенным в первом байте (двух) строки.

Прочие виды информации

Представляемая в ВМ информация может быть статической или динамической [33]. Так, числовая, символьная и логическая информация является статической - ее значение не связано со временем. Напротив, аудиоинформация имеет динамический характер - существует только в режиме реального времени и не может быть остановлена для более подробного изучения. Если изменить масштаб времени, аудиоинформация искажается, что используется, например, для создания звуковых эффектов.

Видеоинформация

Видеоинформация бывает как статической, так и динамической. Статическая видеоинформация включает в себя текст, рисунки, графики, чертежи, таблицы и др. Рисунки делятся также на плоские — двумерные и объемные — трехмерные.

Динамическая видеоинформация - это видео-, мульт- и слайд-фильмы. В их основе лежит последовательное экспонирование на экране в реальном масштабе времени отдельных кадров в соответствии со сценарием. Динамическая информация используется либо для передачи движущихся изображений (анимация), либо для последовательной демонстрации отдельных кадров (слайд-фильмы).

Для демонстрации анимационных и слайд-фильмов опираются на различные принципы. Анимационные фильмы демонстрируются так, чтобы зрительный аппарат человека не мог зафиксировать отдельных кадров (для получения качественной анимации кадры должны сменяться порядка 70 раз/с). При демонстрации слайд-фильмов каждый кадр экспонируется на экране столько времени, сколько необходимо для восприятия его человеком (обычно от 30 с до 1 мин). Слайд-фильмы можно отнести к статической видеоинформации.

В вычислительной технике существует два способа представления графических изображений: *матричный (растровый)* и *векторный*. Матричные (bitmap) форматы хорошо подходят для изображений со сложными гаммами цветов, оттен-

ков и форм, таких как фотографии, рисунки, отсканированные данные. Векторные форматы более приспособлены для чертежей и изображений с простыми формами, тенями и окраской.

В матричных форматах изображение представляется прямоугольной матрицей точек — *пикселов* (picture element), положение которых в матрице соответствует координатам точек на экране. Помимо координат каждый пиксел характеризуется своим цветом, цветом фона или градацией яркости. Количество битов, выделяемых для указания цвета пиксела, изменяется в зависимости от формата. В высококачественных изображениях цвет пиксела описывают 24 битами, что дает около 16 млн цветов. Основной недостаток матричной (растровой) графики заключается в большой емкости памяти, требуемой для хранения изображения, из-за чего для описания изображений прибегают к различным методам сжатия данных. В настоящее время существует множество форматов графических файлов, различающихся алгоритмами сжатия и способами представления матричных изображений, а также сферой применения. Некоторые из распространенных форматов матричных графических файлов перечислены в табл. 2.11.

Таблица **2.11.** Матричные графические форматы

Обозначение	Полное название
BMP	Windows и OS\2 Bitmap
GIF	Graphics Interchange Format
PCX	PC Paintbrush File Format
JPEG	Joint Photographic Experts Group
TIFF	Tagged Image File Format
PNG	Portable Network Graphics

Векторное представление, в отличие от матричной графики, определяет описание изображения не пикселами, а кривыми - сплайнами. Сплайн - это гладкая кривая, которая проходит через две или более опорные точки, управляющие формой сплайна. В векторной графике наиболее распространены сплайны на основе кривых Безье. Суть сплайна: любую элементарную кривую можно построить, зная четыре коэффициента P_0, P_1, P_2 и P_3, соответствующие четырем точкам на плоскости. Перемещение этих точек влечет за собой изменение формы кривой (рис. 2.31).

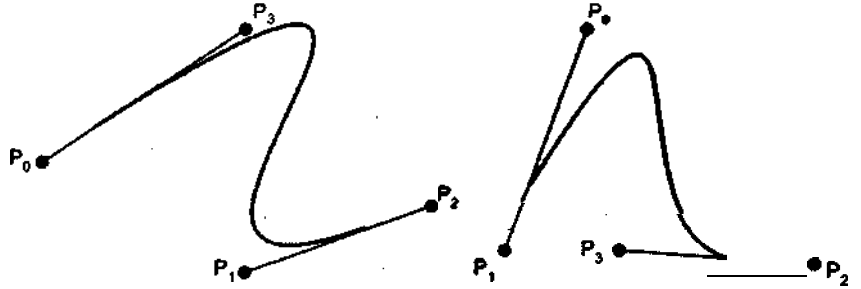


Рис. 2.31. Варианты сплайнов

Хотя это может показаться более сложным, но для многих видов изображений использование математических описаний является более простым способом. В векторной графике для описания объектов используются математические формулы. Это позволяет при рисовании объектов вычислять, куда необходимо помещать реальные точки изображения. Имеется ряд простейших объектов, или *примитивов,* например эллипс, прямоугольник, линия. Эти примитивы и их комбинации служат основой для создания более сложных изображений. В простейшем случае изображение может быть составлено из отрезков линий, для которых задаются начальные координаты, угол наклона, длина, толщина линии, цвет линии и цвет фона.

Основное достоинство векторной графики - описание объекта, является простым и занимает мало памяти. Кроме того, векторная графика в сравнении с матричной имеет следующие преимущества:

- простота масштабирования изображения без ухудшения его качества;
- независимость емкости памяти, требуемой для хранения изображения, от выбранной цветовой модели.

Недостатком векторных изображений является их некоторая искусственность, заключающаяся в том, что любое изображение необходимо разбить на конечное множество составляющих его примитивов. Как и для матричной графики, существует несколько форматов графических векторных файлов. Некоторые из них приведены в табл. 2.12.

Таблица 2.12. Векторные графические форматы

Обозначение	Полное название
DXF	Drawing Interchange Format
CDR	Corel Drawing
HPGL	Hewlett-Packard Graphics Language
PS	PostScript
SVG	Scalable Vector Graphics
VSD	Microsoft Visio format

Матричная и векторная графика существуют не обособленно друг от друга. Так, векторные рисунки могут включать в себя и матричные изображения. Кроме того, векторные и матричные изображения могут быть преобразованы друг в друга. Графические форматы, позволяющие сочетать матричное и векторное описание изображения, *называются метафайлами.* Метафайлы обеспечивают достаточную компактность файлов с сохранением высокого качества изображения.

Таблица 2.13. Форматы метафайлов

Обозначение	Полное название
EPS	Encapsulated PostScript
WMF	Windows Metafile
CGM	Computer Graphics Metafile

Рассмотренные формы представления статической видеоинформации используются, в частности, для отдельных кадров, образующих анимационные фильмы. Для хранения анимационных фильмов применяются различные методы сжатия информации, большинство из которых стандартизовано,.

Аудиоинформация

Понятие *аудио* связано со звуками, которые способно воспринимать человеческое ухо. Частоты аудиосигналов лежат в диапазоне от 15 Гц до 20 КГц, а сигналы по своей природе являются непрерывными (аналоговыми). Прежде чем быть представленной в ВМ, аудиоинформация должна быть преобразована в цифровую форму (оцифрована). Для этого значения звуковых сигналов (выборки, samples), взятые через малые промежутки времени, с помощью аналого-цифровых преобразователей (АЦП) переводятся в двоичный код. Обратное действие выполняется цифро-аналоговыми преобразователями (ЦАП). Чем чаще производятся выборки, тем выше может быть точность последующего воспроизведения исходного сигнала, но тем большая емкость памяти требуется для хранения оцифрованного звука.

Цифровой эквивалент аудиосигналов обычно хранится в виде файлов, причем широко используются различные методы сжатия такой информации. Как правило, к методам сжатия аудиоинформации предъявляется требование возможности восстановления непрерывного сигнала без заметного ухудшения его качества. В настоящее время распространен целый ряд форматов хранения аудиоинформации. Некоторые из них перечислены в табл. 2.14.

Таблица 2.14. Форматы аудиофайлов

Обозначение	Полное название
AVI	Audio Video Interleave
W	WAVeform Extension.
MIDI	Musical Instrument Digital Interface
AIF	Audio Interchange Format
MPEG	Motion Picture Expert Group Audio
RA	Real Audio

Типы команд

Несмотря на различие в системах команд разных ВМ, некоторые основные типы операций могут быть найдены в любой из них. Для описания этих типов примем следующую классификацию:

- команды пересылки данных;
- команды арифметической и логической обработки;
- команды работы со строками;
- команды SIMD;
- команды преобразования;

- команды ввода/вывода;
- команды управления потоком команд.

Команды пересылки данных

Это наиболее распространенный тип машинных команд. В таких командах должна содержаться следующая информация:

- адреса источника и получателя операндов — адреса ячеек памяти, номера регистров процессора или информация о том, что операнды расположены в стеке;
- длина подлежащих пересылке данных (обычно в байтах или словах), заданная явно или косвенно;
- способ адресации каждого из операндов, с помощью которого содержимое адресной части команды может быть пересчитано в физический адрес операнда.

Рассматриваемая группа команд обеспечивает передачу информации между процессором и ОП, внутри процессора и между ячейками памяти. Пересылочные операции внутри процессора имеют тип «регистр-регистр». Передачи между процессором и памятью относятся к типу «регистр-память», а пересылки в памяти — к типу «память-память».

Команды арифметической и логической обработки

В данную группу входят команды, обеспечивающие арифметическую и логическую обработку информации в различных формах ее представления. Для каждой формы представления чисел в АСК обычно предусматривается некий стандартный набор операций.

Помимо вычисления результата выполнение арифметических и логических операций сопровождается формированием в АЛУ признаков (флагов), характеризующих этот результат. Наиболее часто фиксируются такие признаки, как: Z (Zero) - нулевой результат; N (Negative) - отрицательный результат; V (oVerflow) — переполнение разрядной сетки; C (Carry) — наличие переноса.

Операции с целыми числами

К стандартному набору операций над целыми числами, представленными в форме с фиксированной запятой, следует отнести:

- двухместные арифметические операции (операции с двумя операндами): сложение, вычитание, умножение и деление;
- одноместные арифметические операции (операции с одним операндом): вычисление абсолютного значения (модуля) операнда, изменение знака операнда;
- операции сравнения, обеспечивающие сравнение двух целых чисел и выработку признаков, характеризующих соотношение между сопоставляемыми величинами (=, <>, >, <, <=, >=).

Часто этот перечень дополняют такими операциями, как вычисление остатка от целочисленного деления, сложение с учетом переноса, вычитание с учетом заема

увеличение значения операнда на единицу (инкремент), уменьшение значения операнда на единицу (декремент).

Отметим, что *выполнение арифметических* команд может дополнительно сопровождаться перемещением данных из устройства ввода в АЛУ или из АЛУ на устройство вывода.

Операции с числами в форме с плавающей запятой

Для работы с числами, представленными в форме с плавающей запятой, в АСК большинства машин предусмотрены:

- основные арифметические операции: сложение, вычитание, умножение и деление;
- операции сравнения, обеспечивающие сравнение двух вещественных чисел с выработкой признаков: =, <>, >, <, <=, >=;
- операции преобразования: формы представления (между фиксированной и плавающей запятой), формата представления (с одинарной и двойной точностью).

Логические операции

Стандартная система команд ВМ содержит команды для выполнения различных логических операций над отдельными битами слов или других адресуемых единиц. Такие команды предназначены для обработки символьных и логических данных. Минимальный набор поддерживаемых логических операций - это «НЕ», «И», «ИЛИ» и сложение по модулю 2.

Операции сдвигов

В дополнение к побитовым логическим операциям, практически во всех АСК предусмотрены команды для реализации операций логического, арифметического и циклического сдвигов (рис. 2.32).

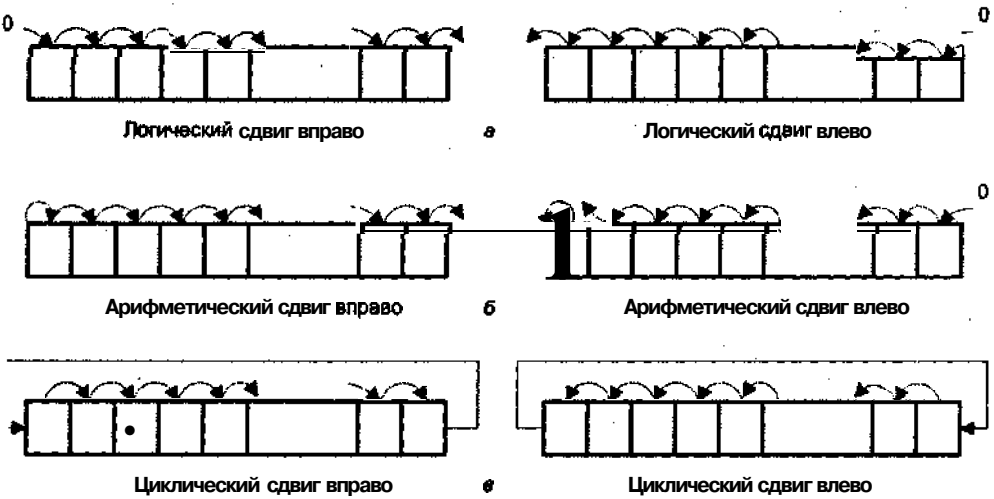


Рис. 2.32. Варианты операций сдвига

При логическом сдвиге влево или вправо (см. рис. 2.32, а), сдвигаются все разряды слова. Биты, вышедшие за пределы разрядной сетки, теряются, а освободившиеся позиции заполняются нулями.

При *арифметическом* сдвиге (см. рис. 2.32, данные трактуются как целые числа со знаком, причем бит знака не изменяет положения. При сдвиге вправо освободившиеся позиции заполняются значением знакового разряда, а при сдвиге влево - нулями. Арифметические сдвиги позволяют ускорить выполнение некоторых арифметических операций. Так, если числа представлены двоичным дополнительным кодом, то сдвиги влево и вправо эквивалентны соответственно умножению и делению на 2.

При *циклическом* сдвиге (см. рис. 2.32, в) смещаются все разряды слова, причем значение разряда, выходящего за пределы слова, заносится в позицию, освободившуюся с противоположной стороны, то есть потери информации не происходит. Одно из возможных применений циклических сдвигов - это перемещение интересующего бита в крайнюю левую (знаковую) позицию, где он может быть проанализирован как знак числа.

Операции с десятичными числами

Десятичные числа представляются в ВМ в двоично-кодированной форме. В вычислительных машинах первых поколений для обработки таких чисел предусматривались специальные команды, обеспечивавшие выполнение основных арифметических операций (сложение, вычитание, умножение и деление). В АСК современных машин подобных команд обычно нет, а соответствующие вычисления имитируются с помощью команд целочисленной арифметики.

SIMD-команды

Название данного типа команд представляет собой аббревиатуру от Single Instruction Multiple Data — буквально «одна инструкция — много данных». В отличие от обычных команд, оперирующих двумя числами, SIMD-команды обрабатывают сразу две группы чисел (в принципе их можно называть групповыми командами). Операнды таких команд обычно представлены в одном из упакованных форматов.

Идея SIMD-обработки была выдвинута в Институте точной механики и вычислительной техники им. С. А. Лебедева в 1978 году в рамках проекта «Эльбрус-1». С 1992 года команды типа SIMD становятся неотъемлемым элементом АСК микропроцессоров фирм Intel и AMD. Поводом послужило широкое распространение мультимедийныхприложений. Видео, трехмерная графика и звук в ВМ представляются большими массивами данных, элементы которых чаще всего обрабатываются идентично. Так, при сжатии видео и преобразовании его в формат MPEG один и тот же алгоритм применяется к тысячам битов данных. В трехмерной графике часто встречаются операции, которые можно выполнить за один такт: интерполирование и нормировка векторов, вычисление скалярного произведения векторов, интерполяция компонентов цвета и т. д. Включение SIMD-команд в АСК позволяет существенно ускорить подобные вычисления.

Первой на мультимедийный бум отреагировала фирма Intel, добавив в систему команд своего микропроцессора Pentium MMX 57 SIMD-команд. Название MMX (MultiMedia eXtention - мультимедийное расширение) разработчики обосновы-

вали тем, что при выборе состава новых команд были проанализированы алгоритмы, применяемые в различных мультимедийных приложениях. Команды MMX обеспечивали параллельную обработку упакованных целых чисел. При выполнении арифметических операций каждое из чисел, входящих в группу, рассматривается как самостоятельное, без связи с соседними числами. Учитывая специфику обрабатываемой информации, команды MMX реализуют так называемую арифметику с насыщением: если в результате сложения образуется число, выходящее за пределы отведенных под него позиций, оно заменяется наибольшим двоичным числом, которое в эти позиции вмещается. На рис. 2.33 показано сложение двух групп четырехразрядных целых чисел, упакованных в 32-разрядные слова.

Рис. 2.33. Сложение с насыщением упакованных целых чисел

Следующим шагом стало создание новых наборов SIMD-команд, работающих также с операндами, представленными в виде упакованных чисел с плавающей запятой. Такие команды в соответствующих приложениях повышают производительность процессора примерно вдвое. Первой подобную технологию в середине 1998 года предложила фирма AMD в микропроцессоре K6-2. Это мультимедийное расширение включало в себя 21 SIMD-команду и получило название 3DNow!. Расширение 3DNow! в дополнение к SIMD-обработке целочисленной информации типа MMX позволяло оперировать парой упакованных чисел в формате с плавающей запятой.

Полугодом позже фирма Intel ввела в свои микропроцессоры так называемые потоковые SIMD-команды, обозначив их аббревиатурой SSE - Streaming SIMD Extension (потоковая обработка по принципу «одна команда - много данных»). Сначала это были 70 команд в микропроцессоре Pentium III. Команды дополняли групповые целочисленные операции MMX и расширяли их за счет групповых операций с 32-разрядными вещественными числами.

В зависимости от типа чисел (целые или вещественные) команды SSE делятся на три категории:

- работа с упакованными группами целых чисел, которые могут иметь размер байта, слова, двойного слова или квадрослова (количество чисел в группе зависит от их разрядности и от разрядности всей группы — 64 или 128);
- оперирование одной парой 32-разрядных или 64-разрядных чисел с плавающей запятой (обычная или двойная точность);

- обработка четырех пар вещественных чисел обычной точности или двух пар вещественных чисел двойной точности.

Дальнейшее развитие технологии SSE вылилось в SSE2 и получило реализацию в Pentium 4. Этот вариант включает в себя 271 команду и позволяет выполнять групповые арифметические и логические операции, сдвиги, сравнения чисел, перегруппировку и извлечение отдельных чисел, различные варианты пересылок. За один такт обрабатываются четыре 32-разрядных числа с плавающей запятой, упакованных в 128-разрядное слово.

Новый импульс получила и технология 3DNow!, более совершенный вариант которой получил название Enhanced 3DNow!. Этот набор команд близок к SSE2.

Таблица 2.15 дает представление о том, какие из рассмотренных мультимедийных расширений поддерживаются наиболее популярными микропроцессорами класса Pentium.

Таблица 2.15. Поддержка мультимедийных расширений в различных микропроцессорах

	MMX	3DNow!	SSE	SSE2
VIA C3	Да	Да	Нет	Нет
Celeron-2, Pentium III	Да	Нет	Да	Нет
Pentium 4	Да	Нет	Да	Да
Duron	Да	Да	Да	Нет
Athlon XP	Да	Да	Да	Нет

Еще один вариант архитектуры системы команд с SIMD-командами воплощен фирмой IBM в процессорах серии PowerPC. Эта реализация носит название AltiVec и во многих отношениях превосходит вышеупомянутые расширения АС К. В частности, имеются трехоперандные команды, допускаются нестандартные целочисленные форматы, например «упаковка» из 1 + 5 + 5 + 5 битов.

Команды для работы со строками

Для работы со строками в АСК обычно предусматриваются команды, обеспечивающие перемещение, сравнение и поиск строк. В большинстве машин перечисленные операции просто имитируются за счет других команд.

Команды преобразования

Команды преобразования осуществляют изменение формата представления данных. Примером может служить преобразование из десятичной системы счисления в двоичную или перевод 8-разрядного кода символа из кодировки ASCII в кодировку EBCDIC, и наоборот.

Команды ввода/вывода

Команды этой группы могут быть подразделены на команды управления периферийным устройством (ПУ), проверки его состояния, ввода и вывода.

Команды *управления периферийным устройством* служат для запуска ПУ и указания ему требуемого действия. Например, накопителю на магнитной ленте мо-

жет быть предписано на необходимость перемотки ленты или ее продвижения вперед на одну запись. Трактовка подобных инструкций зависит от типа ПУ.

Команды *проверки состояния ввода/вывода* применяются для тестирования различных признаков, характеризующих состояние модуля В/ВЫВ и подключенных к нему ПУ. Благодаря этим командам центральный процессор может выяснить, включено ли питание ПУ, завершена ли предыдущая операция ввода/вывода, возникли ли в процессе ввода/вывода какие-либо ошибки и т. п.

Собственно обмен информацией с ПУ обеспечивают команды ввода и вывода. Команды *ввода* предписывают модулю В/ВЫВ получить элемент данных (байт или слово) от ПУ и поместить его на шину данных, а команды *вывода* — заставляют модуль В/ВЫВ принять элемент данных с шины данных и переслать его на ПУ.

Команды управления системой

Команды, входящие в эту группу, являются привилегированными и могут выполняться, только когда центральный процессор ВМ находится в привилегированном состоянии или выполняет программу, находящуюся в привилегированной области памяти (обычно привилегированный режим используется лишь операционной системой). Так, лишь эти команды способны считывать и изменять состояние ряда регистров устройства управления.

Команды управления потоком команд

Концепция фон-неймановской вычислительной машины предполагает, что команды программы, как правило, выполняются в порядке их расположения в памяти. Для получения адреса очередной команды достаточно увеличить содержимое счетчика команд на длину текущей команды. В то же время основные преимущества ВМ заключаются именно в возможности изменения хода вычислений в зависимости от возникающих в процессе счета результатов. С этой целью в АСК вычислительной машины включаются команды, позволяющие нарушить естественный порядок следования и передать управление в иную точку программы. В адресной части таких команд содержится адрес точки перехода (адрес той команды, которая должна быть выполнена следующей). Переход реализуется путем загрузки адреса точки перехода в счетчик команд (вместо увеличения содержимого этого счетчика на длину команды).

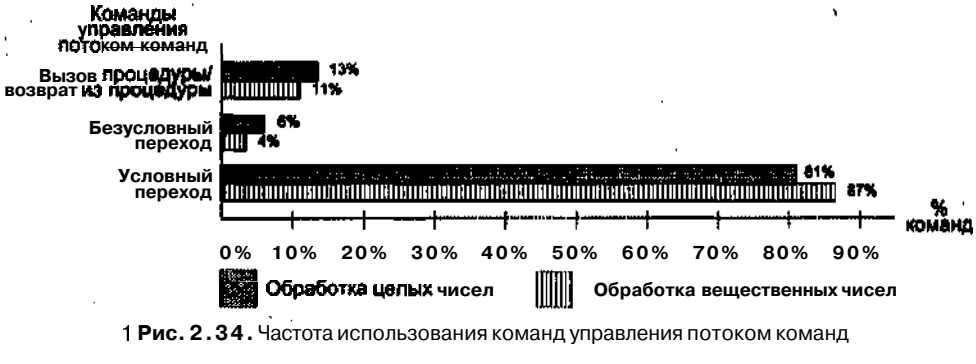
В системе команд ВМ можно выделить три типа команд, способных изменить последовательность вычислений:

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

Степень утилизации каждого из этих типов команд в реальных приложениях иллюстрирует диаграмма, приведенная на рис. 2.34.

Согласно приведенным данным, среди команд рассматриваемой группы доминируют условные переходы.

Несмотря на то что присутствие в программе большого числа *команд безусловного перехода* считается признаком плохого стиля программирования, такие



1 **Рис. 2.34.** Частота использования команд управления потоком команд

команды обязательно входят в АСК любой ВМ. Для их обозначения в языке ассемблера обычно используется английское слово *jump* (прыжок). Команда безусловного перехода обеспечивает переход по заданному адресу без проверки каких-либо условий.

Условный переход происходит только при соблюдении определенного условия, в противном случае выполняется следующая по порядку команда программы. Большинство производителей ВМ в своих ассемблерах обозначают подобные команды словом *branch* (ветвление). Статистика случаев, когда переход имеет место, приведена на рис. 2.35.

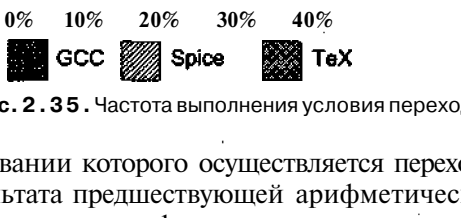


Рис. 2.35. Частота выполнения условия перехода

Условием, на основании которого осуществляется переход, чаще всего выступают признаки результата предшествующей арифметической или логической операции. Каждый из признаков фиксируется в своем разряде регистра флагов процессора. Возможен и иной подход, когда решение о. переходе принимается в зависимости от состояния одного из регистров общего назначения, куда предварительно помещается результат операции сравнения. Третий вариант — это объединение операций сравнения и перехода в одной команде.

В системе команд ВМ для каждого признака результата предусматривается своя команда ветвления (иногда — две: переход при наличии признака и переход при его отсутствии). Большая часть условных переходов связана с проверкой взаимного соотношения двух величин или с равенством (неравенством) некоторой величины нулю. Последний вид проверок используется в программах наиболее интенсивно, о чем свидетельствуют статистические данные, приведенные на рис. 2.36. На верхней диаграмме показаны частоты употребления разных видов условий для программ GCC, Spice и TeX, выполнявшихся на ВМ MIPS 2000. Нижняя диаграмма дает представление о результатах, полученных для двух тестовых смесей

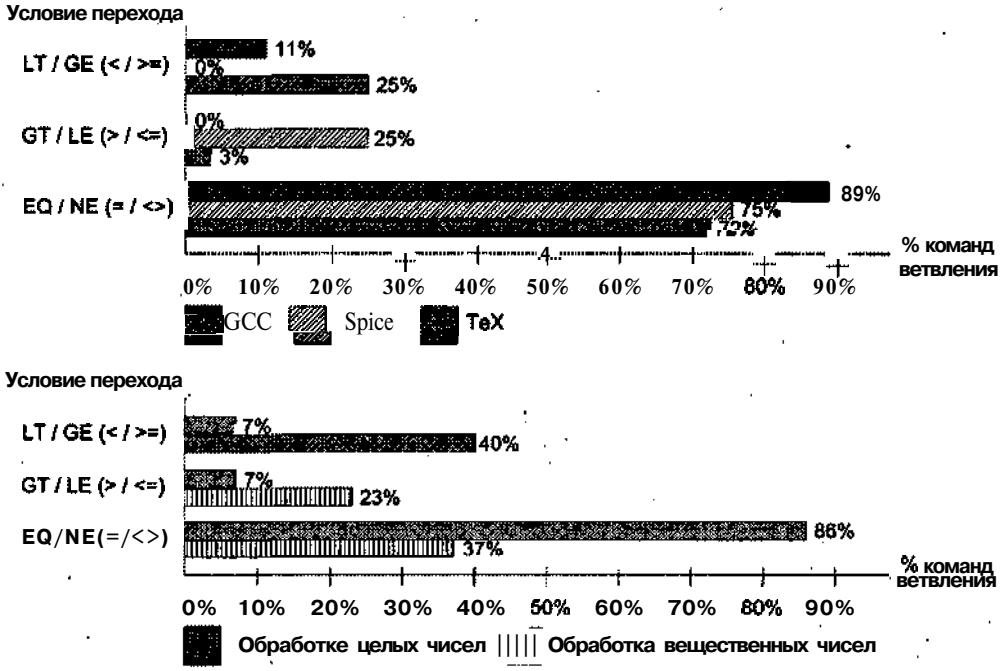


Рис. 2.36. Средняя частота использования различных условий перехода в командах ветвления

программ, где в одной из смесей превалировали целочисленные вычисления, а в другой - вычисления с плавающей запятой.

Помимо приведенных на рис. 2.36 данных необходимо отметить, что по результатам тех же исследований доля команд, где проверяются простые условия (= или <>), составляет в среднем 50%. Кроме того, в качестве одного из операндов в командах, вычисляющих результат сравнения, как правило, выступает константа. Доля таких команд для уже упоминавшихся программ составляет: GCC - 84%, Spice - 92%, TeX - 83%.

Одной из форм команд условного перехода являются *команды пропуска.* В них адрес перехода отсутствует, а при выполнении условия происходит пропуск следующей команды, то есть предполагается, что отсутствующий в команде адрес следующей команды эквивалентен адресу текущей команды, увеличенному на длину пропускаемой команды. Такой прием позволяет сократить длину команд передачи управления.

Для всех языков программирования характерно интенсивное использование механизма процедур. Процедура может быть вызвана в любой точке программы. Для ВМ такой вызов означает, что в этой точке необходимо выполнить процедуру, после чего вернуться в точку, непосредственно следующую за местом вызова.

Процедурный механизм базируется на *командах вызова процедуры,* обеспечивающих переход из текущей точки программы к начальной команде процедуры, и *командах возврата из процедуры,* для возврата в точку, непосредственно распо-

ложенную за командой вызова. Такой режим предполагает наличие средств для сохранения текущего состояния содержимого счетчика команд в момент вызова (запоминание адреса точки возврата) и его восстановления при выходе из процедуры.

Форматы команд

Типовая команда, в общем случае, должна указывать:

- подлежащую выполнению операцию;
- адреса исходных данных (операндов), над которыми выполняется операция;
- адрес, по которому должен быть помещен результат операции.

В соответствии с этим команда состоит из двух частей: операционной и адресной (рис. 2.37).

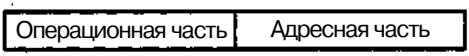


Рис. 2.37. Структура команды

Формат команды определяет ее структуру, то есть количество двоичных разрядов, отводимых под всю команду, а также количество и расположение отдельных полей команды. *Полем* называется совокупность двоичных разрядов, кодирующих составную часть команды. При создании ВМ выбор формата команды влияет на многие характеристики будущей машины. Оценивая возможные форматы, нужно учитывать следующие факторы:

- общее число различных команд;
- общую длину команды;
- тип полей команды (фиксированной или переменной длины) и их длина;
- простоту декодирования;
- адресуемость и способы адресации;
- стоимость оборудования для декодирования и исполнения команд.

Длина команды

Это важнейшее обстоятельство, влияющее на организацию и емкость памяти, структуру шин, сложность и быстродействие ЦП. С одной стороны, удобно иметь в распоряжении мощный набор команд, то есть как можно больше кодов операций, операндов, способов адресации, и максимальное адресное пространство. Однако все это требует выделения большего количества разрядов под каждое поле команды, что приводит к увеличению ее длины. Вместе с тем, для ускорения выборки из памяти желательно, чтобы команда была как можно короче, а ее длина была равна или кратна ширине шины данных. Для упрощения аппаратуры и повышения быстродействия ВМ длину команды обычно выбирают кратной байту, поскольку в большинстве ВМ основная память организована в виде 8-битовых ячеек. В рамках системы команд одной ВМ могут использоваться разные форматы команд. Обычно это связано с применением различных способов адресации. В таком слу-

чае в состав кода команды вводится поле для задания способа адресации (СА), и обобщенный формат команды приобретает вид, показанный на рис. 2.38.

Рис. 2.38. Обобщенный формат команды

Общая длина команды R_K может быть описана следующим соотношением:

$$R_K = \sum_{i=1}^{I} R_{A_i} + R_{КОп} + R_{СА} \quad (2.1)$$

где I — количество адресов в команде; R_{A_i} — количество разрядов для записи i-го адреса; $R_{КОп}$ — разрядность поля кода операции; $R_{СА}$ — разрядность поля способа адресации.

В большинстве ВМ одновременно уживаются несколько различных форматов команд.

Разрядность полей команды

Как уже говорилось, в любой команде можно выделить операционную и адресную части. Длины соответствующих полей определяются различными факторами, которые целесообразно рассмотреть по отдельности.

Разрядность поля кода операции

Количество двоичных разрядов, отводимых под код операции, выбирается так, чтобы можно было представить любую из операций. Если система команд предполагает $N_{КОп}$ различных операций, то минимальная разрядность поля кода операции $R_{КОп}$ определяется следующим образом:

$$R_{КОп} = \text{int}(\log_2 N_{КОп}), \quad (2.2)$$

где int означает округление в оольшую сторону до целого числа.

При заданной длине кода команды приходится искать компромисс между разрядностью поля кода операции и адресного поля. Большее количество возможных операций предполагает длинное поле кода операции, что ведет к сокращению адресного поля, то есть к сужению адресного пространства. Для устранения этого противоречия иногда длину поля кода операции варьируют. Изначально под код операции отводится некое фиксированное число разрядов, однако для отдельных команд это поле расширяется за счет нескольких битов, отнимаемых у адресного поля. Так, например, может быть увеличено число различных команд пересылки данных. Необходимо отметить, что «урезание» части адресного поля ведет к сокращению возможностей адресации, и подобный прием рекомендуется только в тех командах, где подобное сокращение может быть оправданным.

Разрядность адресной части

В адресной части команды содержится информация о местонахождении исходных данных и месте сохранения результата операции. Обычно местонахождение

каждого из операндов и результата задается в команде путем указания адреса соответствующей ячейки основной памяти или номера регистра процессора. Принципы использования информации из адресной части команды определяет *система адресации*. Система адресации задает *число адресов в команде* команды и принятые *способы адресации*.

Разрядности полей R_A и R_{CA} рассчитываются по формулам:

$$R_{A_i} = \text{int}(\log_2 N_i); \quad \textbf{(2.3)}$$

$$R_{CA} = \text{int}(\log_2 N_{CA}), \quad \textbf{(2.4)}$$

где N_i — количество ячеек памяти, к которому можно обратиться с помощью i-го адреса; N_{CA} — количество способов адресации (int означает округление в большую сторону до целого числа).

Количество адресов в команде

Для определения количества адресов, включаемых в адресную часть, будем использовать термин *адресность*. В «максимальном» варианте необходимо указать три компонента: адрес первого операнда, адрес второго операнда и адрес ячейки, куда заносится результат операции. В принципе может быть добавлен еще один адрес, указывающий место хранения следующей инструкции. В итоге имеет место *четырехадресный формат команды* (рис. 2.39). Такой формат поддерживался в ВМ EDVAC, разработанной в 1940-х годах [116].

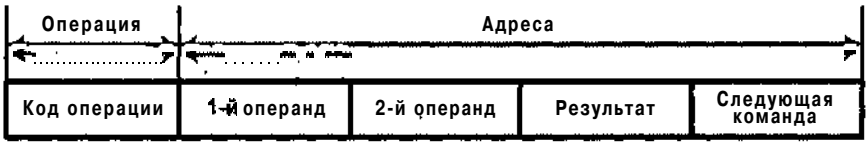


Рис. 2.39. Четырехадресный формат команды

В фон-неймановских ВМ необходимость в четвертом адресе отпадает, поскольку команды располагаются в памяти в порядке их выполнения, и адрес очередной команды может быть получен за счет простого увеличения адреса текущей команды в счетчике команд. Это позволяет перейти к *трехадресному формату команды* (рис. 2.40). Требуется только добавить в систему команд ВМ команды, способные изменять порядок вычислений.

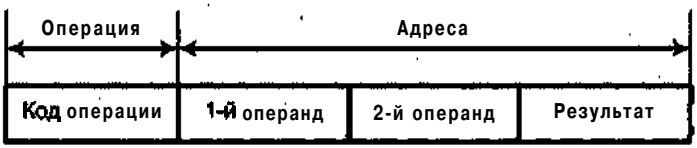


Рис. 2.40. Трехадресный формат команды

К сожалению, и в трехадресном формате длина команды может оказаться весьма большой. Так, если адрес ячейки основной памяти имеет длину 32 бита, а длина кода операции - 8 бит, то длина команды составит 104 бита (13 байт).

Если по умолчанию взять в качестве адреса результата адрес одного из операндов (обычно второго), то можно обойтись без третьего адреса, и в итоге получаем *двухадресный формат команды* (рис. 2.41). Естественно, что в этом случае соответствующий операнд после выполнения операции теряется.

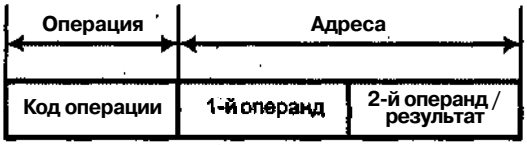


Рис. 2.41. Двухадресный формат команды

Команду можно еще более сократить, перейдя к *одноадресному формату* (рис. 2.42), что возможно при выделении определенного стандартного места для хранения первого операнда и результата. Обычно для этой цели используется специальный регистр центрального процессора (ЦП), известный под названием *аккумулятора,* поскольку здесь аккумулируется результат.

Рис. 2.42. Одноадресный формат команды

Применение единственного регистра для хранения одного из операндов и результата является ограничивающим фактором, поэтому помимо аккумулятора часто используют и другие регистры ЦП. Так как число регистров в ЦП невелико, для указания одного из них в команде достаточно иметь сравнительно короткое адресное поле. Соответствующий формат носит название *полутораадресного* или *регистрового формата* (рис. 2.43).

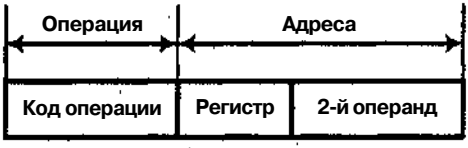


Рис. 2.43. Полутораадресный формат команды

Наконец, если для обоих операндов указать четко заданное местоположение, а также в случае команд, не требующих операнда, можно получить *нульадресный формат команды* (рис. 2.44).

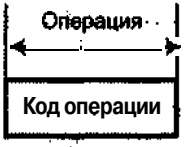


Рис. 2.44. Нульадресный формат команды

В таком варианте адресная часть команды вообще отсутствует или не задействуется.

Выбор адресности команд

При выборе количества адресов в адресной части команды обычно руководствуются следующими критериями:

- емкостью запоминающего устройства, требуемой для хранения программы;
- временем выполнения программы;
- эффективностью использования ячеек памяти при хранении программы.

Для оценки влияния адресности на каждый из перечисленных элементов воспользуемся методикой и выводами, изложенными в [25].

Адресность и емкость запоминающего устройства

Емкость запоминающего устройства для хранения программы E_A можно оценить из соотношения

$$E_A = N_A \times R_{K_A},$$

где N_A - количество программ в программе; R_{K_A} — разрядность команды, определяемая в соответствии с формулой (2.1); A - индекс, указывающий адресность команд программы. С этих позиций оптимальная адресность команды определяется путем решения уравнения $\partial E_A/\partial A=0$ при условии, что найденное значение обеспечивает минимум E_A. В [29] показано, что в среднем E_A монотонно возрастает с увеличением A. Таким образом, *при выборе количества адресов по критерию «емкость* ЗУ» *предпочтение следует отдавать одноадресным командам.*

Адресность и время выполнения программы

Время выполнения одной команды складывается из времени выполнения операции и времени обращения к памяти.

Для трехадресной команды последнее суммируется из четырех составляющих времени:

- выборки команды;
- выборки первого операнда;
- выборки второго операнда;
- записи в память результата.

Одноадресная команда требует двух обращений к памяти:

- выборки команды;
- выборки операнда.

Как видно, на выполнение одноадресной команды затрачивается меньше времени, чем на обработку трехадресной команды, однако для реализации одной трехадресной команды, как правило, нужно три одноадресных. Этих соображений тем

не менее не достаточно, чтобы однозначно отдать предпочтение тому или иному варианту адресности. Определяющим при выборе является тип алгоритмов, на преимущественную реализацию которых ориентирована конкретная ВМ.

В самой общей постановке задачи время выполнения алгоритма T_A можно определить выражением

$$T_A = N_a \tau_{a_A} + N_{\text{н}} \tau_{\text{н}_A}, \tag{2.5}$$

в котором N_a — количество арифметических и логических команд в программе; тета $_{a_A}$-время выполнения однойарифметической или логической команды; $N_{\text{н}}$— количество неарифметических команд; тета На - время выполнения одной неарифметической команды; $A = \{1, 2, 3\}$ - индекс, определяющий количество адресов в команде. В свою очередь, $N_{\text{н}}$ можно определить как $N_{\text{н}} = N_{\text{у}} + {}_{\text{в}_A}$,где $N_{\text{у}}$— количество команд передачи управления (их число в программе не зависит от адресности), а ${}_{\text{в}_A}$ — количество вспомогательных команд пересылок данных в регистр сумматора и из него.

Время выполнения как арифметической (τ_{a_A}), так и неарифметической $(\tau_{\text{н}_A})$ команды складывается из времени выборки команды из памяти $\tau_{\text{о}}$($\tau_{\text{о}}$ —время, затрачиваемое на одно обращение к памяти) и времени считывания/записи данных $A\tau_{\text{о}}$. В случае арифметической команды следует учесть также вклад на исполнение арифметической операции $_a$. Таким образом, имеем:

$$\tau_{a_A} = \tau_a + \tau_{\text{о}} + A\tau_{\text{о}} = \tau_a + (A+1)\tau_{\text{о}}; \tag{2.6}$$

$$\tau_{\text{н}_A} = \tau_{\text{о}} + A\tau_{\text{о}} = (A+1)\tau_{\text{о}}, \tag{2.7}$$

и выражение (2.5) принимает вид:

$$T_A = N_a[\tau_a + (A1)\tau_{\text{о}}] + (N_{\text{у}} + N_{\text{в}_A})(+1)\tau_{\text{о}}. \tag{2.8}$$

Подставляя в (2.8) значения $A = 1$ *и* $A = 3$, можно определить разность времен **дельта Т** еализации алгоритма с помощью одноадресных и трехадресных команд, принимая во внимание, что для трехадресных команд $N_{\text{в}_A} = 0$:

$$AT = T_1 - T_3 = 2\tau_{\text{о}}(N_{\text{в}_1} - N_a - N_{\text{у}}). \tag{2.9}$$

Теперь проанализируем **«выгодность»** той или иной адресности команды в зависимости **от** типа целевого алгоритма. Возможные типы алгоритмов условно разделим на три группы:

- последовательные;
- параллельные;
- комбинированные.

Для последовательного алгоритма результат предшествующей команды используется в последующей. Здесь $N_{\text{в}_1} = 2$, так как требуется всего одна команда предварительной засылки числа в сумматор (аккумулятор) в начале вычисления и одна команда пересылки результата в память в конце вычислений. Если обозначить количество арифметических и логических команд в последовательном алгоритме

и затратами времени T на доступ к адресуемым данным. Затраты оборудования определяются суммой

$$C = C_{ВА} + C_{ЗУ},$$

где $C_{ВА}$ - затраты аппаратных средств, обеспечивающих вычисление исполнительных адресов; $C_{ЗУ}$ — затраты памяти на хранение адресных кодов команд. Обычно $C_{ЗУ} » C_{ВА}$, поэтому при оценке затрат оборудования ограничиваются учетом величины $C_{ЗУ}$. Затраты времени T определяются суммой времени $t_{ФИА}$ формирования исполнительного адреса и времени $t_{ЗУ}$ выборки или записи операнда:

$$T = t_{ФИА} + t_{ЗУ}.$$

В настоящее время используются различные виды адресации, наиболее распространенные из которых рассматриваются ниже.

Непосредственная адресация

При *непосредственной адресации* (НА) в адресном поле команды вместо адреса содержится непосредственно сам операнд (рис. 2.45). Этот способ может применяться при выполнении арифметических операций, операций сравнения, а также для загрузки констант в регистры.

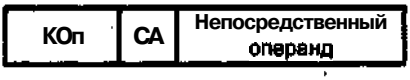


Рис. 2.45. Непосредственная адресация

Когда операндом является число, оно обычно представляется в дополнительном коде. При записи в регистр, имеющий разрядность, превышающую длину непосредственного операнда, операнд размещается в младшей части регистра, а оставшиеся свободными позиции заполняются значением знакового бита операнда.

Помимо того, что в адресном поле могут быть указаны только константы, еще одним недостатком данного способа адресации является то, что размер непосредственного операнда ограничен длиной адресного поля команды, которое в большинстве случаев меньше длины машинного слова. В [120] приведены данные о типичной длине непосредственного операнда для программ GCC, Spice и TeX, выполнявшихся на вычислительной машине DEC VAX (рис. 2.46).

В 50-60% команд с непосредственной адресацией длина операнда не превышает 8 бит, а в 75-80% - 16 бит. Таким образом, в подавляющем числе случаев шестнадцати разрядов вполне достаточно, хотя для вычисления адресов могут потребоваться и более длинные константы.

Рисунок 2.47 дает представление о распространенности непосредственной адресации в командах различных типов. На верхней диаграмме показана статистика для программ GCC, Spice и TeX. Нижняя диаграмма иллюстрирует «популярность» непосредственной адресации в приложениях с преимущественно целочисленными и вещественными вычислениями, однако следует иметь в виду, что средний процент использования непосредственной адресации по всем командам составляет 35% для целочисленных вычислений и 10% — в программах, ориентированных на обработку чисел с плавающей запятой.

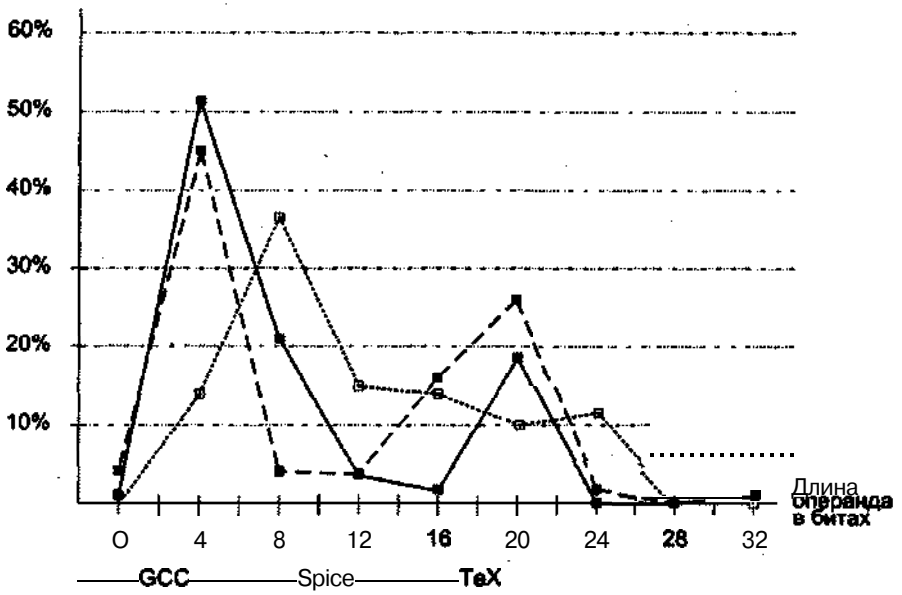


Рис. 2.46. Распределение длин непосредственных операндов для трех программ

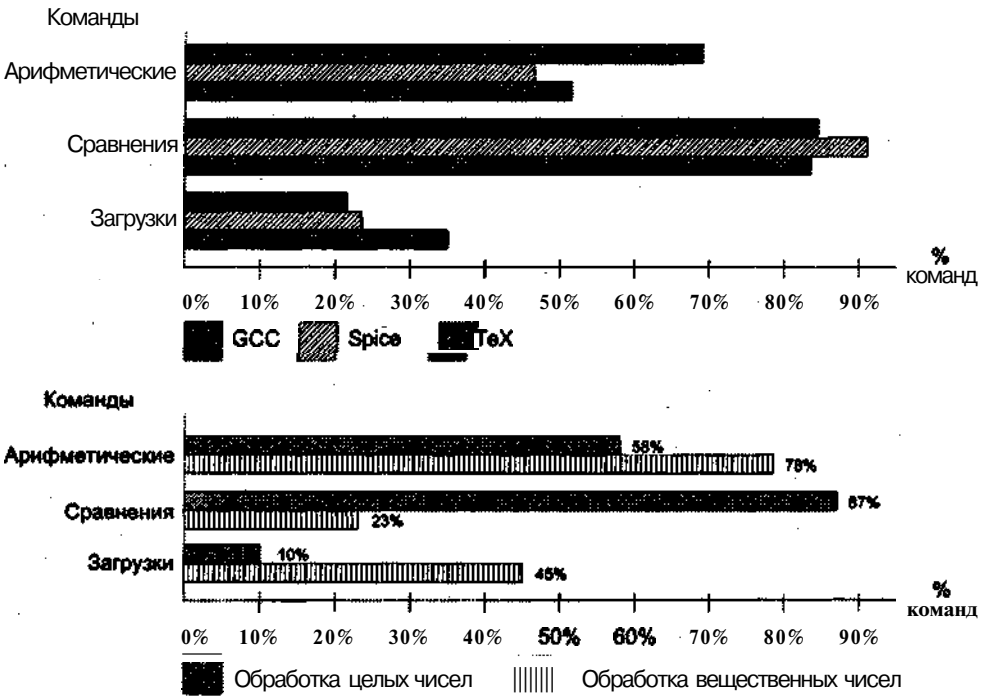


Рис. 2.47. Частота использования непосредственной адресации в командах разных типов

Из приведенных диаграмм видно, что наиболее интенсивно данный вид адресации используется в арифметических операциях и командах сравнения. В то же время загрузка констант в большинстве программ, очевидно, не такая частая операция.

Непосредственная адресация сокращает время выполнения команды, так как не требуется обращение к памяти за операндом. Кроме того, экономится память, поскольку отпадает необходимость в ячейке для хранения операнда. В плане эффективности этот способ можно считать «идеальным» ($C_{НА}$=O, $T_{НА}$=0), и его можно рекомендовать к использованию во всех ситуациях, когда тому не препятствуют вышеупомянутые ограничения.

Прямая адресация

При прямой или абсолютной адресации (ПА) адресный код прямо указывает номер ячейки памяти, к которой производится обращение (рис. 2.48), то есть адресный код совпадает с исполнительным адресом.

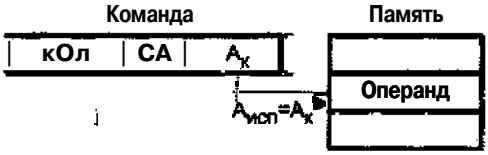


Рис. 2.48. Прямая адресация

При всей простоте использования способ имеет существенный недостаток - ограниченный размер адресного пространства, так как для адресации к памяти большой емкости нужно «длинное» адресное поле. Однако более существенным несовершенством можно считать то, что адрес, указанный в команде, не может быть изменен в процессе вычислений (во всяком случае, такое изменение не рекомендуется) . Это ограничивает возможности по произвольному размещению программы в памяти.

Прямую адресацию характеризуют следующие показатели эффективности: $C_{ПА} = \text{int}(\log_2 N_i)$, $_{ПА} = t_{ЗУ}$, где N_i — количество адресуемых операндов.

Косвенная адресация

Одним из путей преодоления проблем, свойственных прямой адресации, может служить прием, когда с помощью ограниченного адресного поля команды указывается адрес ячейки, в свою очередь, содержащей полноразрядный адрес операнда (рис. 2.49). Этот способ известен как *косвенная адресация* (КА). Запись $(A_К)$ означает содержимое ячейки, адрес которой указан в скобках.

При косвенной адресации содержимое адресного поля команды остается неизменным, в то время как косвенный адрес в процессе выполнения программы можно изменять. Это позволяет проводить вычисления, когда адреса операндов заранее неизвестны и появляются лишь в процессе решения задачи. Дополнительно такой прием упрощает обработку массивов и списков, а также передачу параметров подпрограммам.

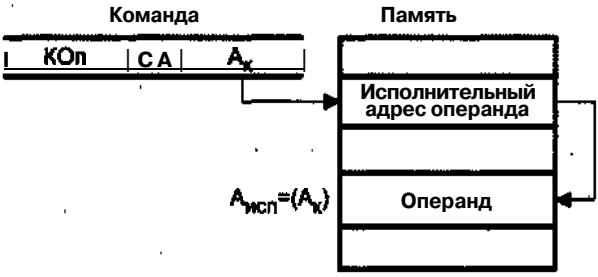


Рис. 2.49. Косвенная адресация

Недостатком косвенной адресации является необходимость в двухкратном обращении к памяти: сначала для извлечения адреса операнда, а затем для обращения к операнду (= $2t_3$). Сверх того задействуется лишняя ячейка памяти для хранения исполнительного адреса операнда. Способу свойственны следующие затраты оборудования:

$$СКА = R_{яч} + \text{int}(\log_2 N_A) = \text{int}(\log_2(+ N_A),$$

где $_{яч}$ — разрядность ячейки памяти, хранящей исполнительный адрес; N_A — количество ячеек для хранения исполнительных адресов; $N_,$ — количество адресуемых операндов. Здесь выражение $\text{int}(\log_2 N_A)$ определяет разрядность сокращенного адресного поля команды (обычно $N_A << N_i$).

В качестве варианта косвенной адресации, правда, достаточно редко используемого, можно упомянуть *многоуровневую* или *каскадную косвенную адресацию:* исп = (... $(A_К)$..•> когда к исполнительному адресу ведет цепочка косвенных адресов. В этом случае один из битов в каждом адресе служит признаком косвенной адресации. Состояние бита указывает, является ли содержимое ячейки очередным адресом в цепочке адресов или это уже исполнительный адрес операнда. Особых преимуществ у такого подхода нет, но в некоторых специфических ситуациях он оказывается весьма удобным, например при обработке многомерных массивов. В то же время очевиден и его недостаток — для доступа к операнду требуется три и более обращений к памяти.

Регистровая адресация

Регистровая адресация (РА) напоминает прямую адресацию. Различие состоит в том, что адресное поле инструкции указывает не на ячейку памяти, а на регистр процессора (рис. 2.50). Идентификатор регистра в дальнейшем будем обозначать буквой *R.* Обычно размер адресного поля в данном случае составляет три или четыре бита, что позволяет указать соответственно на один из 8 или 16 регистров общего назначения (РОН).

Двумя основными преимуществами регистровой адресации являются: короткое адресное поле в команде и исключение обращений к памяти. Малое число РОН позволяет сократить длину адресного поля команды, то есть $C_{РА} << C_{ПА}$. Кроме того, $_{РА} = _{РОН}$, где $_{РОН}$ - время выборки операнда из регистра общего назначения, при-

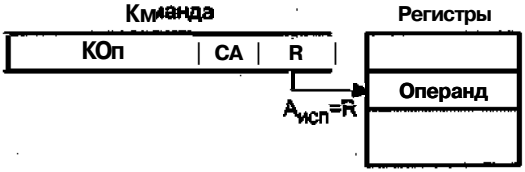


Рис. 2.50. Регистровая адресация

чем $t_{РОН} \ll 2t_{ЗУ}$. К сожалению, возможности по использованию регистровой адресации ограничены малым числом РОН в составе процессора.

Косвенная регистровая адресация

Косвенная регистровая адресация (КРА) представляет собой косвенную адресацию, где исполнительный адрес операнда хранится не в ячейке основной памяти, а в регистре процессора. Соответственно, адресное поле команды указывает не на ячейку памяти, а на регистр (рис. 2.51).

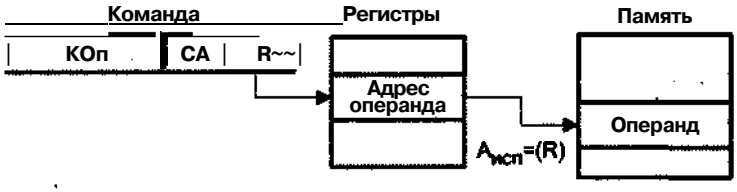


Рис. 2.51. Косвенная регистровая адресация

Достоинства и ограничения косвенной регистровой адресации те же, что и у обычной косвенной адресации, но благодаря тому, что косвенный адрес хранится не в памяти, а в регистре, для доступа к операнду требуется на одно обращение к памяти меньше. Эффективность косвенной регистровой адресации можно оценить по формулам:

$$T_{КРА} = t_{РОН} + t_{ЗУ} \quad C = R_{РОН} + \text{int}(\log_2 N_A) = \text{int}(\log_2(N_i + N_A),$$

где $R_{РОН}$ - разрядность регистров общего назначения.

Адресация со смещением

При адресации со смещением исполнительный адрес формируется в результате суммирования содержимого адресного поля команды с содержимым одного или нескольких регистров процессора (рис. 2.52).

Адресация со смещением предполагает, что адресная часть команды включает в себя как минимум одно поле $(A_К)$. В нем содержится константа, смысл которой в разных вариантах адресации со смещением может меняться. Константа может представлять собой некий базовый адрес, к которому добавляется хранящееся в регистре смещение. Допустим и прямо противоположный подход: базовый адрес находится в регистре процессора, а в поле $A_К$ указывается смещение относительно этого адреса. В некоторых процессорах для реализации определенных вариантов адресации со смещением предусмотрены специальные регистры, например базовый или индексный. Использование таких регистров предполагается по умолча-

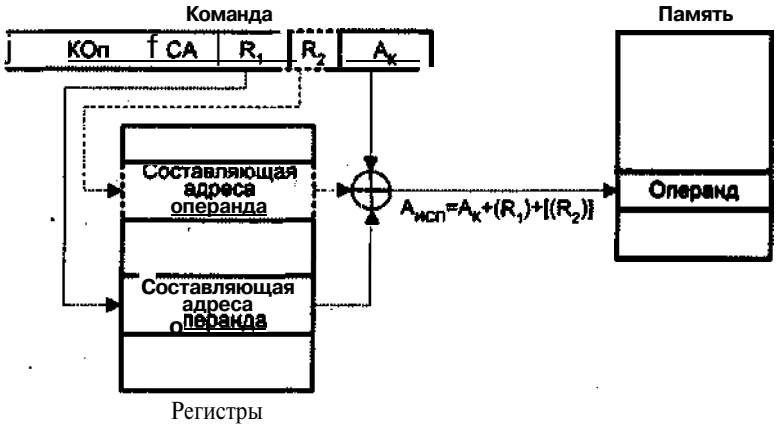


Рис. 2.52. Адресация со смещением

нию, поэтому адресная часть команды содержит только поле $A_к$. Если же составляющая адреса может располагаться в произвольном регистре общего назначения, то для указания конкретного регистра в команду включается дополнительное поле R (при составлении адреса более чем из двух составляющих в команде будет несколько таких полей). Еще одно поле R может появиться в командах, где смещение перед вычислением исполнительного адреса умножается на масштабный коэффициент. Такой коэффициент заносится в один из РОН, на который и указывает это дополнительное поле. В наиболее общем случае адресация со смещением подразумевает наличие двух адресных полей: $A_к$ и R.

В рамках адресации со смещением имеется еще один вариант, при котором исполнительный адрес вычисляется не суммированием, а **конкатенацией** (присоединением) составляющих адреса. Здесь одна составляющая представляет собой старшую часть исполнительного адреса, а вторая — младшую.

Ниже рассматриваются основные способы адресации со смещением, каждый из которых, впрочем, имеет собственное название.

Относительная адресация

При *относительной адресации* (ОА) для получения исполнительного адреса операнда содержимое подполя $A_к$ команды складывается с содержимым счетчика команд (рис. 2.53). Таким образом, адресный код в команде представляет собой смещение относительно адреса текущей команды. Следует отметить, что в момент вычисления исполнительного адреса операнда в счетчике команд может уже быть сформирован адрес следующей команды, что нужно учитывать при выборе величины смещения. Обычно подполе $A_к$ трактуется как двоичное число в дополнительном коде.

Адресация относительно счетчика команд базируется на свойстве локальности, выражающемся в том, Что большая часть обращений происходит к ячейкам, расположенным в непосредственной близости от выполняемой команды. Это позволяет сэкономить на длине адресной части команды, поскольку разрядность подполя $_к$ может быть небольшой. Главное достоинство данного способа адресации состоит в том, что он делает программу перемещаемой в памяти: независимо от

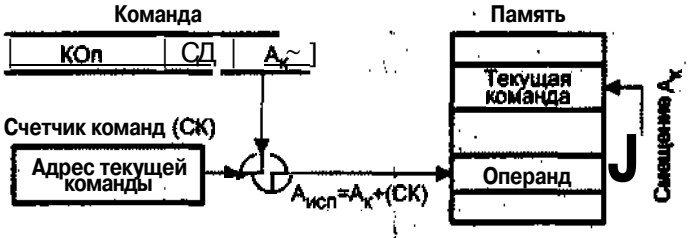


Рис. 2.53. Относительная адресация

текущего расположения программы в адресном пространстве взаимное положение команды и операнда остается неизменным, поэтому адресация операнда остается корректной.

Эффективность данного способа адресации можно описать выражениями:

$$T_C = t_{РОН} + t_{СЛ}; \quad C_{СА} = \text{int}(\log_2 R_{СК}),$$

где $t_{сл}$ — время сложения составляющих исполнительного адреса; $R_{СК}$— разрядность счетчика команд.

Базовая регистровая адресация

В случае *базовой регистровой адресации* (БРА) регистр, называемый базовым, содержит полноразрядный адрес, а подполе A_C—смещение относительно этого адреса. Ссылка на базовый регистр может быть явной или неявной. В некоторых ВМ имеется специальный базовый регистр и его использование является неявным, то есть подполе *R* в команде отсутствует (рис. 2.54).

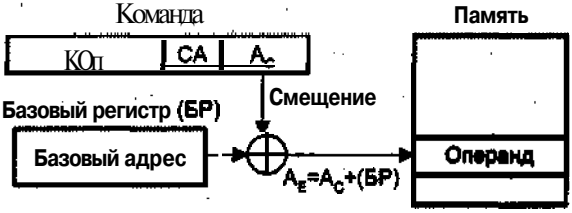


Рис. 2.54. Базовая регистровая адресация с базовым регистром

Более типичен случай, когда в роли базового регистра выступает один из регистров общего назначения (РОН), тогда его номер явно указывается в подполе *R* команды (рис. 2.55).

Базовую регистровую адресацию обычно используют для доступа к элементам массива, положение которого в памяти в процессе вычислений может меняться. В базовый регистр заносится начальный адрес массива, а адрес элемента массива указывается в подполе $_C$ команды в виде смещения относительно начального адреса массива. Достоинство данного способа адресации в том, что смещение имеет меньшую длину, чем полный адрес, и это позволяет сократить длину адресного поля команды. Короткое смещение расширяется до полной длины исполнительного адреса путем добавления слева битов, совпадающих со значением знакового разряда смещения.

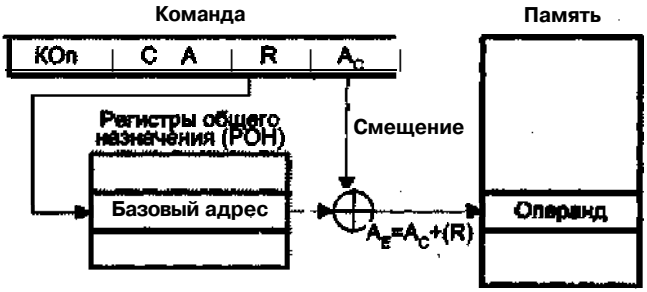


Рис. 2.55. Базовая регистровая адресация с использованием одного из РОН

Распределение значений смещения, оцененное в [120] при выполнении пятитестовых программ пакета SPECInt92 и пяти программ пакета SPECftp92, показано на рис. 2.56.

Рис. 2.56. Распределение размеров смещений при базовой регистровой адресации

Длина смещения превысила 16 бит лишь в 1% случаев базовой регистровой адресации, то есть в подавляющем большинстве случаев длина смещения существенно меньше.

Разрядность смещения $R_{СМ}$ и, соответственно, затраты оборудования определяются из условия $R_{СМ} = C_{БРА} = \text{int}(\log_2(\max(N_{ОП})))$, где $N_{ОП}$ — количество операндов в i-ой программе. С оставляют: $T_Б = t_{РОН} + t_{СЛ} + t_{ЗУ}$.

Индексная адресация

При *индексной адресации* (ИА) подполе содержит адрес ячейки памяти, а регистр (указанный явно или неявно) — смещение относительно этого адреса. Как видно, этот способ адресации похож на базовую регистровую адресацию. Поскольку при индексной адресации в поле A_C находится полноразрядный адрес ячейки памяти, играющий роль базы, длина этого поля больше, чем при базовой регистровой адресации. Тем не менее вычисление исполнительного адреса операнда производится идентично (рис. 2.57, 2.58).

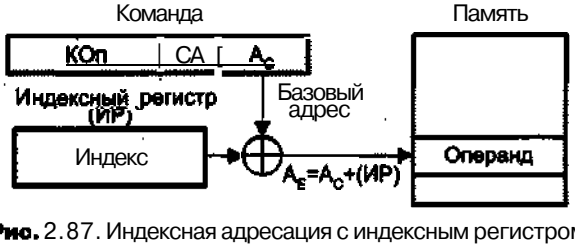


Рис. 2.87. Индексная адресация с индексным регистром

Рис. 2.58. Индексная адресация с использованием одного из РОН

Индексная адресация предоставляет удобный механизм для организации итеративных вычислений. Пусть, например, имеется массив чисел, расположенных в памяти последовательно, начиная с адреса *N,* и мы хотим увеличить на единицу все элементы данного массива. Для этого требуется извлечь каждое число из памяти, прибавить к нему 1 и вернуть обратно, а последовательность исполнительных адресов будет следующей: *N, N*+ 1, *N*+ 2 и т. д., вплоть до последней ячейки, занимаемой рассматриваемым массивом. Значение *N* берется из подполя $_C$команды, а в выбранный регистр, называемый *индексным регистром,* сначала заносится *0.* После каждой операции содержимое индексного регистра увеличивается на 1.

. Так как это довольно типичный случай, в большинстве ВМ увеличение или уменьшение содержимого индексного регистра до или после обращения к нему осуществляется автоматически как часть машинного цикла. Такой прием называется *автоиндексированием.* Если для индексной адресации используются специально выделенные регистры, автоиндексирование может производиться неявно и автоматически. При задействовании для хранения индексов регистров общего назначения необходимость операции автоиндексирования должна указываться в команде специальным битом.

Автоиндексирование с увеличением содержимого индексного регистра носит название *автоинкрементной адресации* и может быть описано следующим образом:

$$\text{Лисп } -A_C+ (R), R \leftarrow (R) + 1 \text{ или } R(R) + 1, A_{\text{исп}} -A_C+ (\text{Л}).$$

В первом варианте увеличение содержимого индексного регистра происходит после формирования исполнительного адреса, и этот способ называется *постинкрементным автоиндексированием.* Во втором случае сначала производится увеличение содержимого индексного регистра, и уже новое значение используется для

формирования исполнительного адреса. Тогда говорят о *преинкрементном автоиндексировании*.

Автоиндексирование с уменьшением содержимого индексного регистра носит название *автодекрементной адресации* и может быть описано так:

$$A_{\text{исп}} = A_C + (R), R \leftarrow (R) - 1 \text{ или } R \leftarrow (R) - 1, A_{\text{исп}} = A_C + (\text{Л}).$$

Здесь также возможны два варианта, отличающиеся последовательностью выполнения операций уменьшения содержимого индексного регистра и вычисления исполнительного адреса: *постдекрементное автоиндексирование* и *предекрементное автоиндексирование*.

Интересным и весьма полезным является еще один вариант индексной адресации — *индексная адресация с масштабированием и смещением:* содержимое индексного регистра умножается на масштабный коэффициент и суммируется с A_C. Масштабный коэффициент может принимать значения 1, 2, 4 или 8, для чего в адресной части команды выделяется дополнительное поле. Описанный способ адресации реализован, например, в микропроцессорах фирмы Intel.

Следует особо отметить, что система команд многих ВМ предоставляет возможность различным образом сочетать базовую и индексную адресации в качестве дополнительных способов адресации.

Страничная адресация

Страничная адресация (СТА) предполагает разбиение адресного пространства на страницы. Страница определяется своим начальным адресом, выступающим в качестве базы. Старшая часть этого адреса хранится в специальном регистре *—регистре адреса страницы* (РАС). В адресном коде команды указывается смещение внутри страницы, рассматриваемое как младшая часть исполнительного адреса. Исполнительный адрес образуется конкатенацией (присоединением) A_{CK} содержимому РАС, как показано на рис. 2.59. На рисунке символ ‖ обозначает операцию конкатенации.

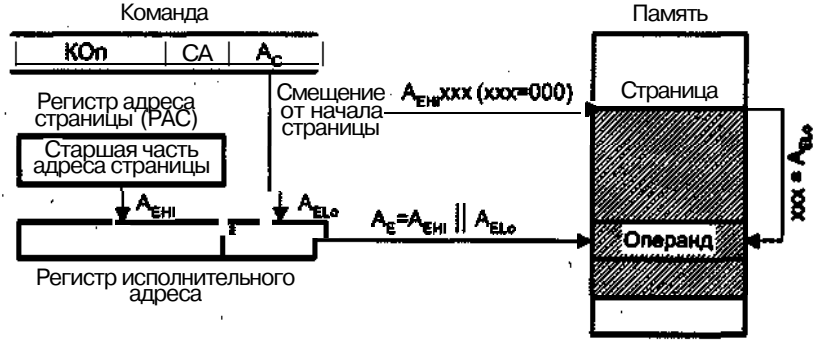


Рис. 2.59. Страничная адресация

Показатели эффективности страничной адресации имеют вид:

$$C_{\text{СТА}} - \text{int}(\log_2 N_i - \log_2 M), T_{\text{СТА}} - T_{\text{РОН}} + T_{\text{ЗУ}},$$

где M — количество страниц в памяти.

Блочная адресация

Блочная адресация используется в командах, для которых единицей обработки служит блок данных, расположенных в последовательных ячейках памяти. Этот способ очень удобен при работе с внешними запоминающими устройствами и в операциях с векторами. Для описания блока обычно берется адрес ячейки, где хранится первый или последний элемент блока, и общее количество элементов блока, заданное числом байтов или ячеек. Вместо длины блока может использоваться специальный признак «конец блока», помещаемый за последним элементом блока

Стековая адресация

Данный вид адресации был рассмотрен при описании стековой архитектуры системы команд.

Распространенность различных видов адресации

Частота использования различных способов адресации существенно зависит от типа АСК. Для машин со стековой архитектурой очевидно, что основным способом адресации является стековая адресация. Для ВМ с аккумуляторной АСК главные способы адресации — это прямая и непосредственная.

Достаточно ясна и ситуация с RISC-архитектурой. Из самой идеи этого подхода вытекает, что преимущественный способ адресации здесь - регистровая адресация.

Более сложным является вопрос о частоте использования различных видов адресации в регистровых ВМ. В рамках этой архитектуры существует множество машин с самыми разнообразными списками команд и различными сочетаниями способов адресации, в силу чего дать однозначный ответ относительно наиболее распространенных вариантов практически невозможно. Сказанное подтверждают результаты, полученные при выполнении программ GCC и Spice на вычислительной машине DEC VAX (рис. 2.60) и на ВМ с микропроцессором класса 80x86 (рис. 2.61).

Из диаграмм видно, что в машине VAX из применявшихся способов адресации доминируют непосредственная, базовая регистровая и косвенная регистровая. Доля не упомянутых в таблице способов адресации не превышает 2%.

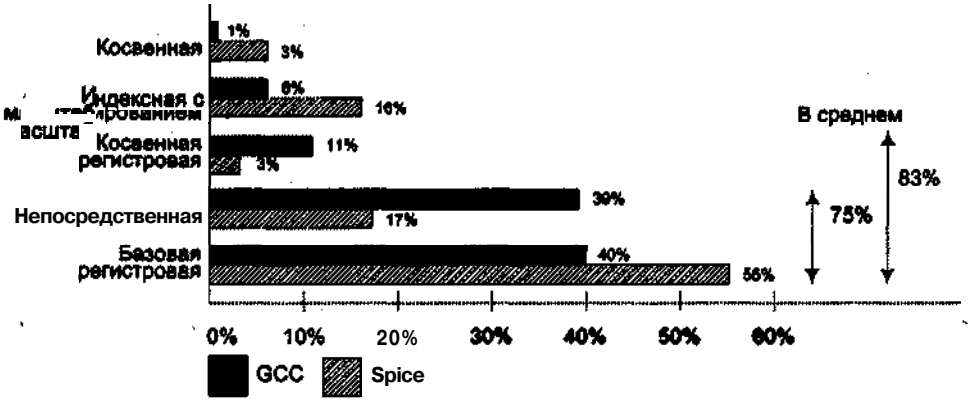


Рис. 2.60. Частота использования методов адресации на программах GCC и Spice (DEC VAX)

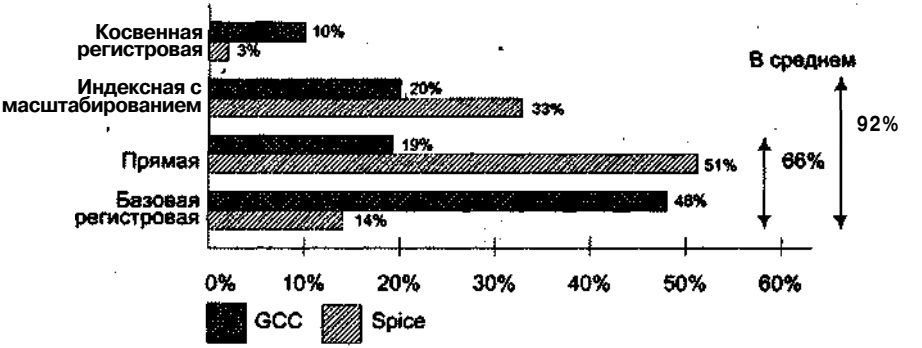


Рис. **2.61.** Частота использования методов адресации на программах GCC и Spice (Intel **80x86)**

При выполнении программ CC и Spice на ВМ с микропроцессорами серии 80x86 наиболее активно используются прямая и базовая регистровая адресации.

Как видно, сделать однозначный вывод о наибольшей распространенности какого-либо способа адресации для архитектур с РОН достаточно сложно. Единственное общее замечание - интенсивность применения конкретных способов адресации ощутимо зависит от характера решаемой задачи. Это обстоятельство обязательно должно учитываться пользователями при выборе ВМ под конкретное применение.

Способы адресации в командах управления потоком команд

Основными способами адресации в командах управления потоком команд являются прямая и относительная.

Для команд безусловного и условного перехода (ветвления) наиболее типична относительная адресация, когда в адресной части команды указывается смещение адреса точки перехода относительно текущей команды, то есть смещение относительно текущего содержимого счетчика команд. Использование данного способа адресации позволяет программе выполняться в любом месте памяти — программы становятся перемещаемыми. Среди команд безусловного перехода доля относительной адресации составляет около 90%.

Для команд перехода чрезвычайно важно, насколько далеко адрес перехода отстоит от адреса команды перехода, иными словами, какова типичная величина смещения. В [120] приведены данные о типовой величине смещения, оцененной по программам GCC, Spice, TeX; они представлены на рис. 2.62. Результаты, полученные на смесях программ с преимущественной обработкой целочисленных и вещественных данных, показаны на нижнем графике того же рисунка.

Как видно, длина смещения в основном не превышает 8 бит, что соответствует смещению в пределах ±128 относительно команды ветвления. В подавляющем большинстве случаев переход идет в пределах 3-7 команд относительно команды перехода.

Рисунок 2.63 дает представление о преимущественном направлении переходов.

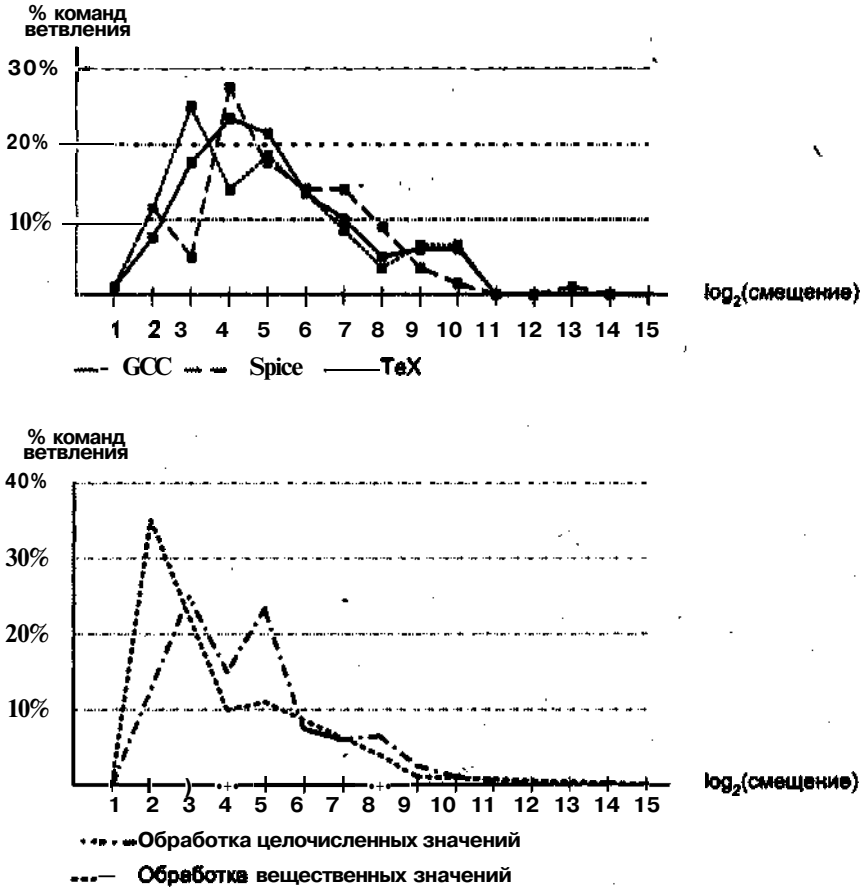


Рис. 2.62. Средние значения смещения в командах условного перехода

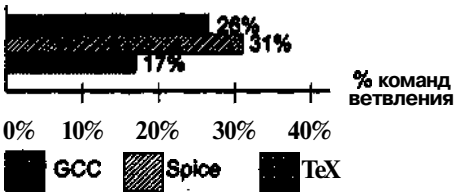


Рис. 2.63. Доля переходов назад в программах GCC, Spice и TeX

Из приведенных данных следует, что в среднем 75% переходов происходит в направлении увеличения адреса. Из переходов в сторону уменьшения адреса около 90% связаны с выполнением циклов.

Система операций

Системой операций называется список операций, непосредственно выполняемых техническими средствами вычислительной машины. Система операций ВМ опре-

деляется областью ее применения, требованиями к стоимости, производительности и точности вычислений.

Связь системы операций с алгоритмами решаемых задач проявляется в степени ее приспособленности для записи программ реализации этих алгоритмов. Степень приспособленности характеризуется близостью списка операций системы команд и операций, используемых на каждом шаге выполнения алгоритмов. Простоту программирования алгоритма часто определяют термином «программируемость вычислительной машины». Чем меньше команд требуется для составления программы реализации какого-либо алгоритма, тем программируемость выше. В архитектурах типа CISC улучшения Программируемо добиваются введением в систему операций большого количества операций, в том числе и достаточно сложных. Это может приводить и к повышению производительности ВМ, хотя в любом случае увеличивает аппаратурные затраты.

Обоснованный выбор системы операций (СО) возможен лишь исходя из анализа подлежащих реализации алгоритмов. Для этого определяется частотный вектор используемых в алгоритме операторов $(q_1, ..., q_n)$. Изучив вектор, составляют список основных, наиболее часто встречающихся операторов. Операторы основного списка реализуются системой машинных операций ВМ (каждому оператору сопоставляется своя машинная операция). Остальные операторы получают путем их разложения на операторы основного списка.

Показатели эффективности системы операций

Качество системы операций можно характеризовать двумя свойствами: функциональной полнотой и эффективностью.

Функциональная полнота - это достаточность системы операций для описания любых алгоритмов. Системы операций ВМ включают в себя большое количество машинных операций и практически всегда являются функционально полными.

Эффективность системы операций показывает степень соответствия СО заданному классу алгоритмов и требованиям к производительности ВМ. Количественно эффективность характеризуется затратами оборудования, затратами времени на реализацию алгоритмов и вероятностью правильного выполнения программ.

Затраты оборудования C можно описать выражением.

$$C = C_{ПР} + C_{ЗУ},$$

где $C_{ПР}$ — затраты в процессоре на реализацию системы операций, $C_{ЗУ}$ — затраты памяти на размещение данных и программ, представляющих алгоритм в терминах заданной системы операций.

Величина $C_{ПР}$ пропорциональна количеству и сложности машинных операций, а $C_{ЗУ}$ — емкости памяти, необходимой для хранения закодированного алгоритма. Усложнение машинных операций приводит к сокращению Количества операций (команд), требуемых для описания алгоритма, и, следовательно, к уменьшению необходимой емкости памяти.

Затраты времени на реализацию алгоритма (Т) пропорциональны количеству команд (операций) в программе. Введение в СО более сложных операций позволяет программировать сложные действия одной командой, в результате чего уменьшается количество команд программы.

Вероятность правильного выполнения программ P определяется по формуле

$$P = \prod_{i=1}^{l} p_i = \prod_{i=1}^{l} e^{-\lambda_i n q_i \tau_i}, \qquad (2.13)$$

где l — количество операций в СО; n - количество операций (команд) в программах; лямбда i, тета i - интенсивность отказов и время выполнения операции (команд типа; q_i — частота появления операций i-го типа в программах; p_i— вероятность правильного выполнения nq_i операций i-го типа.

В [16] утверждается, что показатель P оптимален при условии

$$P_i = P_j, \ (i, j = 12, \ \ldots \ /). \qquad (2.14)$$

Условие (2.14) можно переписать в ином виде:

$$\lambda_i q_i \tau_i = \lambda_j q_j \tau_j, \ (i, j = 1, \ 2, \ \ldots, \ l). \qquad (2.15)$$

Равенство (2.15) позволяет найти оптимальное соотношение параметров лямбда i, при реализации системы операций. Принимая параметры аппаратной реализации одной из операции за эталон $(\lambda_{эт}, \tau_{эт})$, из (2.15) получим соотношение

$$\lambda_i \tau_i = \frac{q_{эт} \lambda_{эт} \tau_{эт}}{Яг}, \ (i=1, \ 2, \ \ldots, \ l), \qquad (2.16)$$

позволяющее по известным параметрам для 5 различных вариантов системы операций выбрать лучший. Критерием качества реализации каждой операции для разработки 5-го варианта СО является

$$\min_{s \in S} \Delta_{i,s} = \min \left| \frac{q_{эт} \lambda_{эт} \tau_{эт}}{q_i} - \lambda_{i,s} \tau_{i,s} \right| \ 0 = 1, \ \ldots \ l). \qquad (2.17)$$

При выборе одного из нескольких вариантов реализации системы операций критерием качества может служить минимум среднеквадратического отклонения

$$\sigma = \min_{s \in S} \left(\sqrt{\sum_{i=1}^{l} \Delta_{i,s}^2} \right). \qquad (2.18)$$

Если значения λ_i (i - 1,..., l) неизвестны, то оправдано следующее допущение: $\lambda_i = \lambda_j$, $((i, j = 1, 2, \ldots,$ОХ

При этом условие (2.15) преобразуется в условие «равного временного участия» операций в программах [28]:

$$q_i \tau_i = q_j \tau_j (i, j = 1, \ 2, \ \ldots, \ l). \qquad (2.19)$$

Условие (2.19) совместно с рассчитанным частотным вектором и известным ограничением на время выполнения программы $T_{доп}$ можно применить для вычисления оптимальной длительности машинных операций:

$$\tau_i = \frac{T_{доп}}{q_i \times n \times l}. \qquad (2.20)$$

Иерархия систем операций

Пусть задан некоторый класс алгоритмов Л, и для его описания используется функционально полная система операций F_1 ,содержащая любые математические операции (интегрирование, ln, *d/dt,* sin, ...). Многие сложные операции с помощью численных методов могут быть представлены в терминах элементарных операций. Последовательно применяя процедуру такого разложения сложных операций на простые, можно построить новые системы операций F_2, , ..., F_p, также функционально полные по отношению к классу алгоритмов *Л.* Системы операций F_1 ..., F_p можно упорядочить по сложности входящих в них операций так, чтобы для каждой пары F_1 и F_{i+1} выполнялось условие: каждая операция f_g из $_i$ либо входит в F_{i+} либо представляется композицией операций/$_h$ из F_{i+1}. Упорядоченная последовательность F_1 ..., F_p образует иерархию систем операций.

Зависимость показателей эффективности /для иерархии систем операций $F.,.., F_p$ представлена на рис. 2.64. Кривая $C_{ПР}$ характеризует затраты оборудования ВМ в зависимости от состава операций $F_i(i = 1, ...,p)$. Поскольку операции в F_{i+} менее сложны, чем операции в F_i, то затраты $C_{ПР}$ процессора, реализующего F_{i+1}, будут меньше затрат $C_{ПР_i}$ процессора, реализующего F_i ,то есть иерархия систем опе-

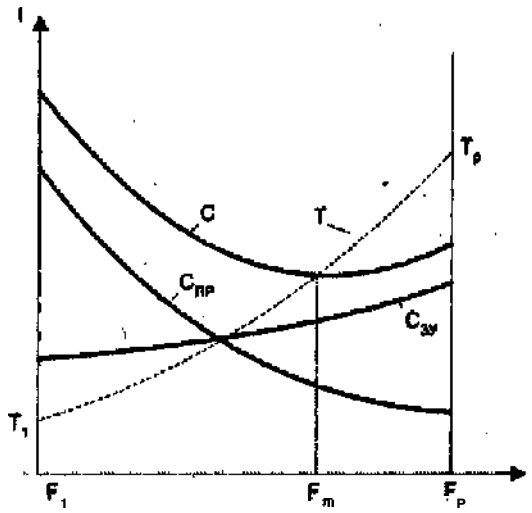


Рис. 2.64. Показатели эффективности иерархии систем операций

раций F_1 ..., F_p соответствует иерархии процессоров с аппаратными затратами С ПР1 > $C_{ПР_2}$ >- > С ПРр

Затратам памяти на рис. 2.64 соответствует кривая $C_{ЗУ}$. Она показывает, что если кодированию алгоритма в терминах операций F_i адекватны затраты памяти $C_{ЗУ_i}$, то использование более простых операций F_{i+} приводит к увеличению количества команд в программе и $C_{ЗУ_{i+1}} > C_{ЗУ_i}$, то есть иерархия систем операций F_1, F_p имеет следствием иерархию затрат памяти $C_{ЗУ} < C_{ЗУ_2} < ... < C_{ЗУ_p}$. Характер изменения суммарных затрат оборудования $C = C_{ПР} + C_{ЗУ}$ представлен кривой C. Видно, что существует система операций F_m, которой соответствуют минимальные затраты оборудования C. Система операций F_m занимает промежуточное положение между наиболее сложной F_1 и наименее сложной F_p системами операций. Наконец, кривая T демонстрирует, что время выполнения алгоритмов возрастает от минимального значения Г, (при наиболее сложной СО F_1) до максимального значения T_p (при наименее сложной СО F_p).

Таким образом, снижение сложности операций в иерархии F_1,..., F_p вызывает:

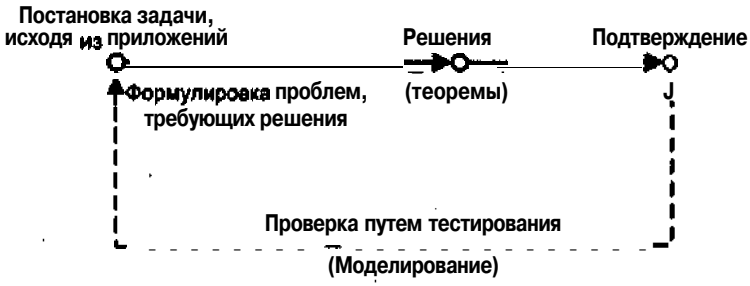
- уменьшение аппаратных затрат в процессоре;
- увеличение затрат памяти для хранения программ;
- увеличение времени реализации алгоритмов (убавление производительности ВМ).

Последний вывод можно считать справедливым лишь для архитектур типа CISC. По отношению к RISC-архитектурам, где сокращение системы операций сопровождается коренными изменениями в других аспектах АСК, подобное утверждение представляется весьма спорным.

Выбор системы операций

Выбор оптимальной системы операций является сложной задачей. Основные трудности в ее решении связаны с установлением точной функциональной зависимости показателей эффективности C, F, P от состава СО. Поэтому найти чисто формальный метод выбора оптимальной СО пока не удается, а существующие подходы к ее решению основываются на комбинации формальных и эвристических приемов. Используемые в настоящее время методы разработки системы команд можно классифицировать следующим образом:

- На основе существующих аналогов, применяемых для решения задач данного класса.
- На основе статистики использования отдельных команд в «старых» вычислительных машинах. Подобная статистика уже заложена во многие общепризнанные контрольно-оценочные тесты, такие, например, как смесь Гибсона или SPEC.
- Сориентацией на языки программирования высокого уровня. Выбор системы операций направлен на реализацию типовых операторов языков программирования. Подобные попытки можно встретить в вычислительных машинах фирм Wang, Hewlett-Packard, Tektronix, IBM.
- На основе формализации и систематизации. Сущность этого метода поясняет рис. 2.65.



Форматы команд 121

Рис. 2.65. Метод формализации и систематизации

- На базе метода группировки. Для упрощения декодирования и повышения производительности все операции группируются (арифметические, логические, передачи управления и т. д.). Выбор состава операций во многом зависит от принципа группировки и возможности извлечения из нее реальных выгод.
- На основе восходящей совместимости в рамках семейства вычислительных машин. Новая система операций должна быть надмножеством старой системы, причем пересечение множеств должно быть существенным.
- Опираясь на требования пользователя.
- На основе метода условной интерпретации. С учетом соотношения «стоимость/производительность» определенные команды реализуются аппаратно или микропрограммно. Как интерпретация команд, так и метод их реализации выбираются из условий полезности данных команд, стоимости и производительности.

В качестве примеров рассмотрим два способа решения задачи выбора оптимальной системы операций, соответствующие упомянутым выше принципам.

Выбор системы операций на основе существующих аналогов

Заданы алгоритмы *A*. Необходимо выбрать функционально полную систему операций, удовлетворяющую ограничениям на время реализации алгоритмов *A* ($T = T_{доп}$), и аппаратную сложность ВМ ($C = C_{доп}$).

Сначала определяется множество ВМ-аналогов, успешно применяемых для выполнения алгоритмов, близких к заданным [17], для которого составляются полный *F* и базовый $F_Б$ наборы операций. Полный набор операций $F\{f_2 ..., f_n\}$ является объединением систем операций всех машин-аналогов, то есть F объединение $\bigcup_{i=1}^{k} F_i$ от 1 до k, где *к* - количество аналогов. В него могут быть включены дополнительные операции, наличие которых необходимо по тем или иным соображениям. Базовый набор $F_Б = \{f_1, f_2 ..., f_m\}$ включает в себя только те операции набора *F,* которые реализованы в большинстве аналогов, то есть здесь $m<n$. Полный набор F соответствует максимально возможной системе операций, а базовый набор $F_Б$ - минимально необходимой СО.

Поиск «оптимальной» СО осуществляется в интервале от $F_Б$ до F. В качестве начального приближения выбирается набор $F_Б$. Производится оценочный расчет реализуемости СО в пределах заданных ограничений ($T_{доп}$, $C_{доп}$), функциональных возможностей и функциональной независимости системы операций.

Требуемая длительность τ_i^T операций определяется исходя из $T_{доп}$ по формуле (2.20). Поиск оптимального варианта реализации каждой из операций осуществляется по критерию $\min \Delta_i = \min |\tau_i^T - \tau_i|$, где τ_i - реальная длительность операции, а для всего множества операций - по критерию минимума среднеквадратического отклонения

$$\sigma = \min_{s \in S} \left(\sqrt{\sum_{i=1}^{m} \left| \tau_i^T - \tau_{i,s} \right|^2} \right),$$

где S — множество рассматриваемых вариантов реализации СО.

Далее проверяется выполнение заданных ограничений $(T_{доп}, C_{доп})$. При их выполнении набор $F_Б$ пополняется операцией из $\bar{F}$ (для улучшения функциональных возможностей СО), в противном случае - сокращается. Выбор операций для корректировки $F_Б$ осуществляется следующим образом. Если принять, что полный набор F обеспечивает эффективную реализацию алгоритмов, то функциональные возможности $F_Б$ можно оценивать длиной D программы из набора операций $F_Б$, интерпретирующей операции из набора F, не попавшие в $F_Б$ (операции набора $\bar{F} = F - F_Б$).

Функциональная независимость операций $\bar{F}_Б$ оценивается количеством операций или шагов C_{f_i} программы описания каждой из операции $f_i \in F_Б$ $(i = 1, 2, ..., m)$ через другие операции этого набора $(F_Б - f_i)$ $(i = 1, 2, ..., m)$. Показатели C_{f_i} $(i = 1, 2, ..., m)$ образуют вектор с длиной m. При сокращении $F_Б$ из-за невыполнения ограничения $C \le C_{доп}$ из набора исключается операция, которая наиболее просто выражается через другие операции $F_Б$ (имеет минимальное значение C_{f_i}), что приводит к минимальным потерям производительности (минимальному увеличению D). При расширении $F_Б$ из-за невыполнения ограничения $T = T_{доп}$ (или из-за необходимости улучшить функциональные возможности СО) в наоор включается операция, обеспечивающая максимальное уменьшение времени T (максимальное уменьшение показателя D). Если таких операций в наборе F несколько, то из них выоирается операция с наименьшим значением так как ее реализация требует минимальных аппаратных затрат.

Описанная последовательность действий повторяется многократно до получения системы операций, удовлетворяющей заданным ограничениям.

Выбор системы операций на основе структурирования алгоритмов

Метод структурирования алгоритмов предполагает следующую формулировку задачи выбора системы операций: необходимо выбрать такую систему F_n, которая обеспечивает реализацию алгоритма A за заданное время $T = T_{доп}$ при минимальной стоимости ВМ.

Иерархия операций $F_1, ..., F_p$, функционально полных для алгоритма A, может быть определена процедурой структурирования [22], сводящейся к следующему. Алгоритм A рассматривается как один оператор, реализующий операцию f_1 над исходными данными с целью получения требуемых результатов, то есть $F_1 = \{f_1\}$. Затем оператор разделяется на части - программируется последовательностью более простых операторов. Последовательное применение процедуры структурирования (разделения оператора на более простые операторы) позволяет выявить

системы операций F_1 - $\{f_1\}$, F_2 - $\{f_2, f_3\}$, F_3 - $\{f_3, f_4, f_5\}$, F_4 = $\{f_4, f_5, f_6$, Л). -, и тем самым построить иерархию операций $F_1, \ldots, F_p$.

Для каждой операции в F_i можно определить количество n_g ее выполнений при одной реализации алгоритма. Тогда сумма

$$N_i = \sum_{f_g \in F_i}^{G} n_g \qquad (2.21)$$

будет представлять количество операций, выполняемых при одной реализации алгоритма, запрограммированного в терминах F_i. Характеристики элементной базы позволяют задать приближенное значение средней длительности т, операции в ВМ. С учетом этого время выполнения алгоритма на основе F_i составит $T_i = \tau$, что дает возможность поставить в соответствие иерархии систем операций $F_1 \ldots F_p$ затраты времени на реализацию алгоритма $T_1 \ldots, T_p$, причем $T_1 < \ldots < T_p$. Можно предположить, что минимум аппаратных затрат достигается при F_n принадлежит $\{F_1 \ldots F_p\}$, обеспечивающей время реализации алгоритма T_n, максимально близкое к заданному значению $T_{доп}$. В силу сказанного, выбор системы операций сводится к нахождению такой системы F_n, для которой разность $(T_{доп} - T_n)$ имеет минимальное положительное значение.

Контрольные вопросы

1. Какие характеристики вычислительной машины охватывает понятие «архитектура системы команд»?
2. Охарактеризуйте эволюцию архитектур системы команд вычислительных машин.
3. В чем состоит проблема семантического разрыва?
4. Поясните различия в подходах по преодолению семантического разрыва, применяемых в ВМ с CISC- и RISC-архитектурами.
5. Какая форма записи математических выражений наиболее соответствует стековой архитектуре системы команд и почему?
6. Какие средства используются для ускорения доступа к вершине стека в ВМ со стековой архитектурой?
7. Чем обусловлено возрождение интереса к стековой архитектуре?
8. Какие особенности аккумуляторной архитектуры можно считать ее достоинствами и недостатками?
9. Какие доводы можно привести за и против увеличения числа регистров общего назначения в ВМ с регистровой архитектурой системы команд?
10. Почему для ВМ с RISC-архитектурой наиболее подходящей представляется АСК с выделенным доступом к памяти?
11. Какую позицию запятой в формате с фиксированной запятой можно считать общепринятой?
12. Чем в формате с фиксированной запятой заполняются избыточные старшие разряды?

13. Какое минимальное количество полей должен содержать формат с плавающей запятой?
14. Как в формате с плавающей запятой решается проблема работы с порядками, имеющими разные знаки?
15. В чем состоит особенность трактовки нормализованной мантиссы в стандарте IEEE 754?
16. От чего зависят точность и диапазон представления чисел в формате с плавающей запятой?
17. Чем обусловлено появление форматов с упакованными числами в современных микропроцессорах?
18. Какие факторы влияют на выбор разрядности целых чисел?
19. Сказывается ли на производительности ВМ порядок следования в памяти байтов «длинного» числа и выбор адреса, с которого начинается запись числа?
20. По какому признаку при передаче потока десятичных чисел можно определить окончание одного числа и начало следующего?
21. Какой общий принцип лежит в основе различных таблиц кодировки символов?
22. Чем обусловлен переход от кодировки ASCII к кодировке Unicode?
23. В чем состоит особенность обработки логических данных?
24. Какие трактовки включает в себя понятие «строка»?
25. Перечислите способы представления графической информации и охарактеризуйте особенности каждого из них.
26. Каким образом в вычислительной машине представляется аудиоинформация?
27. Какой вид команд пересылки данных характерен для ВМ с RISC-архитектурой?
28. Чем вызвана необходимость заполнения освободившихся разрядов значением знакового разряда при арифметическом сдвиге вправо?
29. В чем состоит особенность SIMD-команд и в каком формате должны быть представлены операнды?
30. Что такое «арифметика с насыщением» и где она применяется?
31. Какие виды команд относят к командам ввода/вывода?
32. Какие виды команд условного перехода обычно доминируют в реальных программах?
33. Какие факторы определяют выбор формата команд?
34. Перечислите возможные пути сокращения длины кода команды.
35. Какая особенность фон-неймановской архитектуры позволяет отказаться от указания в команде адреса очередной команды?
36. Какие факторы необходимо учитывать при выборе оптимальной адресности команд?
37. С какими ограничениями связано использование непосредственной адресации?

38. В каких случаях может быть удобна многоуровневая косвенная адресация?
39. Какие преимущества дает адресация относительно счетчика команд?
40. В чем проявляются сходство и различия между базовой и индексной адресацией?
41. В чем состоит сущность автоиндексирования и в каких ситуациях оно применяется?
42. С какой целью применяется адресация с масштабированием?
43. Какие способы адресации переходов используются в Командах управления потоком команд?
44. Как можно оценить эффективность системы операций при разработке архитектуры системы команд?
45. Охарактеризуйте основные методики проектирования и оценки системы команд.

Глава 3
Функциональная организация фон-неймановской ВМ

Данная глава посвящена рассмотрению базовых принципов построения и функционирования фон-неймановских вычислительных машин.

Функциональная схема фон-неймановской вычислительной машины

Чтобы получить более детальное представление о структуре и функциях устройств ВМ, обратимся к схеме гипотетической машины с аккумуляторной архитектурой (рис. 3.1). Для упрощения изложения приняты следующие характеристики машины:

- **Одноадресные команды.** Адресная часть команды содержит только один адрес. При выполнении операций с двумя операндами предполагается, что другой операнд находится в специальном регистре АЛУ — аккумуляторе, а результат также остается в аккумуляторе.
- **Единство форматов.** Длина команд и данных совпадает с разрядностью ячеек памяти, то есть любая команда или операнд занимают только одну ячейку памяти. Таким образом, адрес очередной команды в памяти может быть получен путем прибавления единицы к адресу текущей команды, а для извлечения из памяти любой команды или любого операнда достаточно одного обращения к памяти.

На функциональной схеме (см. рис. 3.1) показаны типовые узлы каждого из основных устройств ВМ, а также сигналы, инициирующие выполнение отдельных операций по пересылке информации и ее обработке, необходимых для функционирования машины.

Устройство управления

Назначение устройства управления (УУ) было определено ранее при рассмотрении структурной схемы ВМ, где отмечалось, что эта часть ВМ организует автоматическое выполнение программ и функционирование ВМ как единой системы. Теперь остановимся на описании узлов, реализующих целевую функцию УУ.

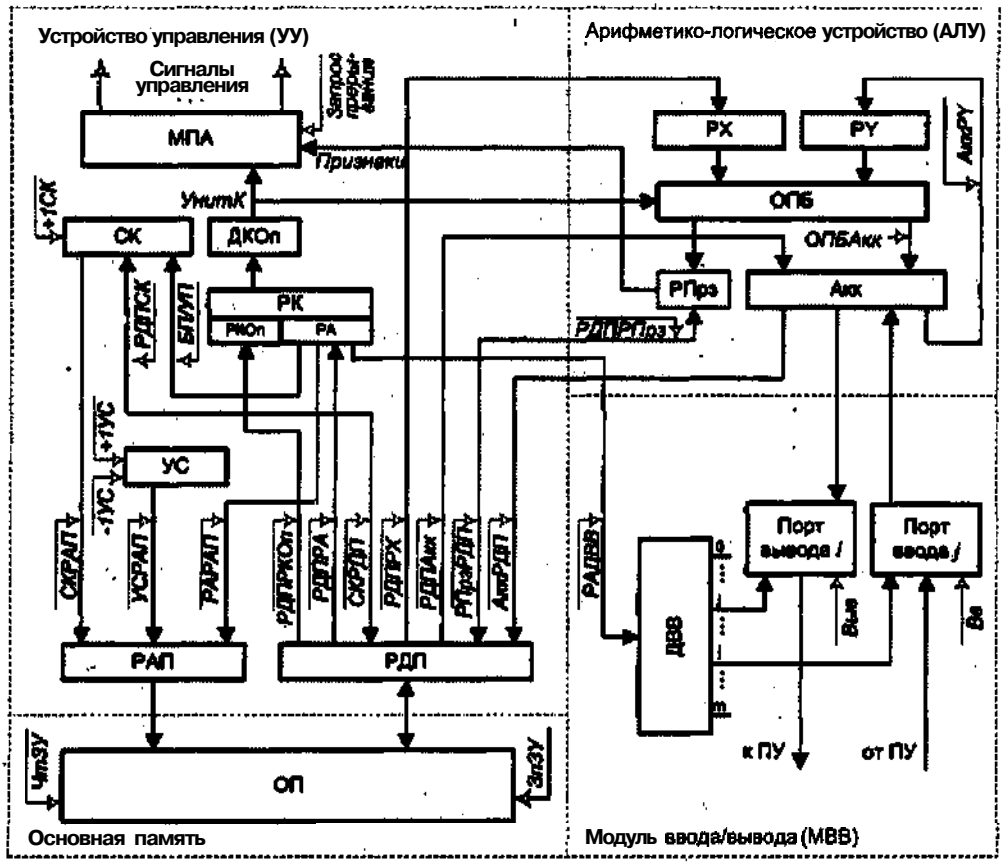


Рис. 3.1.Функциональная схема гипотетической фон-неймановской ЭВМ

Счетчик команд

Счетчик команд (СК) - неотъемлемый элемент устройства управления любой ВМ, построенной в соответствии с фон-неймановским принципом программного управления. Согласно этому принципу соседние команды программы располагаются в ячейках памяти со следующими по порядку адресами и выполняются преимущественно в той же очередности, в какой они размещены в памяти ВМ. Таким образом, адрес очередной команды может быть получен путем увеличения адреса ячейки, из которой была считана текущая команда, на длину выполняемой команды, представленную числом занимаемых ею ячеек. Реализацию такого режима и при-

зван обеспечивать счетчик команд — двоичный счетчик, в котором хранится и модифицируется адрес очередной команды программы. Перед началом вычислений в СК заносится адрес ячейки основной памяти, где хранится команда, которая должна быть выполнена первой. В процессе выполнения каждой команды путем увеличения содержимого СК на длину выполняемой команды в счетчике формируется адрес следующей подлежащей выполнению команды. В рассматриваемой ВМ любая команда занимает одну ячейку, поэтому содержимое СК увеличивается на единицу, что обеспечивается подачей сигнала управления +1СК По завершении текущей команды адрес следующей команды программы всегда берется из счетчика команд. Для изменения естественного **порядка вычислений** (перехода в иную точку программы) достаточно занести в СК адрес точки перехода.

Хотя термин **«счетчик** команд» считается общепринятым, его нельзя признать вполне удачным из-за того, что он создает неверное впечатление о задачах данного узла. По этой причине разработчики ВМ используют иные названия, в частности *программный счетчик* (PC, Program Counter) или *указатель команды* (IP, Instruction Pointer). Последнее определение представляется наиболее удачным, поскольку точнее отражает назначение рассматриваемого узла УУ.

В заключение добавим, что в ряде ВМ счетчик команд реализуется в виде обычного регистра, а увеличение его содержимого производится внешней схемой (схемой инкремента/декремента).

Регистр команды

Счетчик команд определяет лишь местоположение команды в памяти, но не содержит информации о том, что это за команда. Чтобы приступить к выполнению команды, ее необходимо извлечь из памяти и **разместить в** *регистре команды* (РК). Этот этап носит *название выборки команды.* Только с момента загрузки команды в **РК** она становится «видимой» для процессора. В РК команда хранится в течение всего времени ее выполнения. Как уже отмечалось ранее, любая команда содержит два поля: поле кода операции и поле адресной части. Учитывая это обстоятельство, регистр команды иногда рассматривают как совокупность двух регистров -*регистра кода операции* (РКОп) *ирегистраадреса* (РА), в которых хранятся соответствующие составляющие команды.

Если команда занимает несколько последовательных ячеек, то код операции всегда находится в том слове команды, которое извлекается из памяти первым. Это позволяет по коду операции определить, требуются ли считывание из памяти и загрузка в РК остальных слов команды. Собственно выполнение команды начинается только после занесения в РК ее полного кода.

Указатель стека

Указатель стека (УС) - это регистр, где хранится адрес вершины стека. В реальных вычислительных машинах стек реализуется в виде участка основной памяти, обычно расположенного в области наибольших адресов. Заполнение стека происходит в сторону **уменьшения** адресов, при этом вершина стека — это ячейка, куда была произведена последняя по времени запись. Для хранения адреса такой ячейки и предназначен УС. При выполнении операции *push* (занесение в стек) содер-

жимое УС с помощью сигнала -1УС сначала уменьшается на единицу, после чего используется в качестве адреса, по которому производится запись. Соответствующая ячейка становится новой вершиной стека. Считывание из стека (операция *рор)* происходит из ячейки, на которую указывает текущий адрес в УС, после чего содержимое указателя стека сигналом +1УС увеличивается на единицу. Таким образом, вершина стека опускается, а считанное слово считается удаленным из стека. Хотя физически считанное слово и осталось в ячейке памяти, при следующей записи в стек оно будет заменено новой информацией.

Регистр адреса памяти

Регистр адреса памяти (РАП) предназначен для хранения адреса ячейки основной памяти вплоть до завершения операции (считывание или запись) с этой ячейкой. Наличие РАП позволяет компенсировать различия в быстродействии ОП и прочих устройств машины.

Регистр данных памяти

Регистр данных памяти (РДП) призван компенсировать разницу в быстродействии запоминающих устройств и устройств, выступающих в роли источников и потребителей хранимой информации. В РДП при чтении заносится содержимое ячейки ОП, а при записи — помещается информация, подлежащая сохранению в ячейке ОП. Собственно момент считывания и записи в ячейку определяется сигналами ЧтЗУ и ЗпЗУ соответственно.

Дешифратор кода операции

Дешифратор кода операции (ДКОп) преобразует код операции в форму, требуемую для работы микропрограммного автомата (МПА). Информация после декодирования определяет последующие действия МПА, а ее вид зависит от организации МПА. В рассматриваемой ВМ - это унитарный код УнитК Часто код операции преобразуется в адрес первой команды микропрограммы, реализующей указанную в команде операцию. С этих позиций ДКОп правильнее было бы назвать не дешифратором, а преобразователем кодов.

Микропрограммный автомат

Микропрограммный автомат (МПА) правомочно считать центральным узлом устройства управления. Именно МПА формирует последовательность сигналов управления, в соответствии с которыми производятся все действия, необходимые для выборки из памяти и выполнения команд. Исходной информацией для МПА служат: декодированный код операции, состояние признаков (флагов), характеризующих результат предшествующих вычислений, а также внешние запросы на прерывание текущей программы и переход на программу обслуживания прерывания.

Арифметико-логическое устройство

Это устройство, как следует из его названия, предназначено для арифметической и логической обработки данных. В машине, изображенной на рис. 3.1, оно содержит следующие узлы.

Операционный блок

Операционный блок (ОПБ) представляет собой ту часть АЛУ, которая, собственно, и выполняет арифметические и логические операции над поданными на вход операндами. Выбор конкретйой операции из возможного списка операций для данного ОПБ определяется кодом операции команды. В нашей ВМ код операции поступает непосредственно из регистра команды. В реальных машинах КОп зачастую преобразуется в МПА в иную форму и уже из микропрограммного автомата поступает в АЛУ. Операционные блоки современных АЛУ строятся как комбинационные схемы, то есть они не обладают внутренней памятью и до момента сохранения результата операнды должны присутствовать на входе блока.

Регистры операндов

Регистры РХ и РУ обеспечивают сохранение операндов на входе операционного блока вплоть до получения результата операции и его записи (в нашем случае в аккумулятор).

Регистр признаков

Регистр признаков (РПрз) предназначен для фиксации и хранения признаков (флагов), характеризующих результат последней выполненной арифметической или логической операции. Такие признаки могут информировать о равенстве результата нулю, о знаке результата, о возникновении переноса из старшего разряда, переполнении разрядной сетки и т. д. Содержимое РПрз обычно используется устройством управления для реализации условных переходов по результатам операций АЛУ. Под каждый из возможных признаков отводится один разряд РПрз.

Формирование признаков осуществляется блоком формирования состояний регистра признаков, который может входить в состав ОПБ либо реализуется в виде внешней схемы, располагаемой между операционным блоком и РПрз.

Аккумулятор

Аккумулятор (Акк) - это регистр, на который возлагаются самые разнообразные функции. Так, в него предварительно загружается один из операндов, участвующих в арифметической или логической операции. В Акк может храниться результат предыдущей команды и в него же заносится результат очередной операции. Через Акк зачастую производятся операции ввода и вывода.

Строго говоря, аккумулятор в равной мере можно отнести как к АЛУ, так и к УУ, а в ВМ с регистровой архитектурой его можно рассматривать как один из регистров общего назначения.

Основная память

Вне зависимости от типа используемых микросхем основная память (ОП) представляет собой массив запоминающих элементов (ЗЭ), организованных в виде ячеек, способных хранить некую единицу информации, обычно один байт. Каждая ячейка имеет уникальный адрес. Ячейки ОП организованы в виде матрицы, а выбор ячейки осуществляется путем подачи разрешающих сигналов на соответствующие строку и столбец этой матрицы. Это обеспечивается дешифратором

адреса памяти, преобразующим поступивший из РАП адрес ячейки в разрешающие сигналы, подаваемые в горизонтальную и вертикальную линии, на пересечении которых расположена адресуемая ячейка. При современной емкости ОП для реализации данных сигналов приходится использовать несколько микросхем запоминающих устройств (ЗУ). В этих условиях процесс обращения к ячейке состоит из выбора нужной микросхемы (на основании старших разрядов адреса) и выбора ячейки внутри микросхемы (определяется младшими разрядами адреса). Первая часть процедуры производится внешними схемами, а вторая — внутри микросхем ЗУ.

Модуль ввода/вывода

Структура приведенного на рис. 3.1 *модуля ввода/вывода* (МВБ) обеспечивает только пояснение логики работы ВМ. В реальных ВМ реализация этого устройства машины может существенно отличаться от рассматриваемой. Задачей МВБ является обеспечение подключения к ВМ различных периферийных устройств (ПУ) и обмена информацией с ними. В рассматриваемом варианте МВВ состоит из дешифратора номера порта ввода/вывода, множества портов ввода и множества портов вывода.

Порты ввода и порты вывода

Портом называют схему, ответственную за передачу информации из периферийного устройства ввода в аккумулятор АЛУ (порт ввода) или из аккумулятора на периферийное устройство вывода (порт вывода). Схема обеспечивает электрическое и логическое сопряжение ВМ с подключенным к нему периферийным устройством.

Дешифратор номера порта ввода/вывода

В модуле ввода/вывода рассматриваемой ВМ предполагается, что каждое ПУ подключается к своему порту. Каждый порт имеет уникальный номер, который указывается в адресной части команд ввода/вывода. *Дешифратор номера порта ввода/вывода* (ДВВ) обеспечивает преобразование номера порта в сигнал, разрешающий операцию ввода или вывода на соответствующем порте. Непосредственно ввод (вывод) происходит при поступлении из МПА сигнала Вв (Выв).

Микрооперации и микропрограммы

Для пояснения логики функционирования ВМ ее целесообразно представить в виде совокупности узлов, связанных между собой коммуникационной сетью (рис. 3.2).

Процесс функционирования вычислительной машины состоит из последовательности пересылок информации между ее узлами и элементарных действий, выполняемых в узлах. Понятие узла здесь трактуется весьма широко: от регистра до АЛУ или основной памяти. Также широко следует понимать и термин «элементарное действие». Это может быть установка регистра в некоторое состояние или выполнение операции в АЛУ. Любое элементарное действие производится при поступлении соответствующего сигнала управления (СУ) из микропрограммного автомата устройства управления. Возможная частота формирования сигналов на

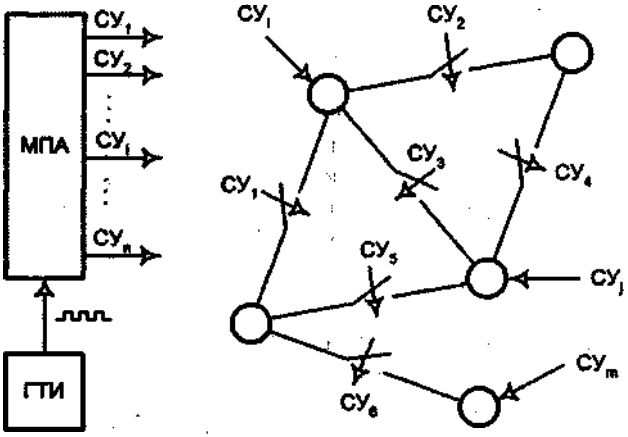


Рис. 3.2. Вычислительнаямашина с позиций микроопераций и сигналов управления

выходе автомата определяется синхронизирующими импульсами, поступающими от генератора тактовых импульсов (ГТИ). Элементарные пересылки или преобразования информации, выполняемые в течение одного такта сигналов синхронизации, называются *микрооперациями*. В течение одного такта могут одновременно выполняться несколько микроопераций. Совокупность сигналов управления, вызывающих микрооперации, выполняемые в одном такте, называют *микрокомандой*. Относительно сложные действия, осуществляемые вычислительной машиной в процессе ее работы, реализуются как последовательность микроопераций и могут быть заданы последовательностью микрокоманд, называемой *микропрограммой*. Реализует микропрограмму, то есть вырабатывает управляющие сигналы, задаваемые ее микрокомандами, *микропрограммный автомат* (МПА).

Способы записи микропрограмм

Для записи микропрограмм в компактной форме используются граф-схемы алгоритмов и языки микропрограммирования.

Граф-схемы алгоритмов

Граф-схема алгоритма (ГСА) имеет вид ориентированного графа. При построении графа оперируют пятью типами вершин (рис. 3.3).

Начальная вершина (см. рис. 3.3, *а*) определяет начало микропрограммы и не имеет входов. Конечная вершина (см. рис. 3.3, *б*) указывает конец микропрограммы, по-

Рис. 3.3, Разновидности вершин граф-схемы алгоритма: *а* — начальная; *б* — конечная; *в* — операторная; *г* — условная; *д* — ждущая

этому имеет только вход. В операторную вершину (см. рис. 3.3, в) вписывают микрооперации, выполняемые в течение одного машинного такта. С вершиной связаны один вход и один выход. Условная* вершина (см. рис. 3.3, *г)* используется для ветвления вычислительного процесса. Она имеет один вход и два выхода, соответствующие позитивному («Да») и негативному («Нет») исходам проверки условия, записанного в вершине. С помощью ждущей вершины (см. рис. 3.3, *д)* можно описывать ожидание в работе устройств. В этом случае выход «Да» соответствует снятию причины, вызвавшей ожидание.

Граф-схемы алгоритмов составляются в соответствии со следующими правилами:

1. ГСА должна содержать одну начальную, одну конечную и конечное множество операторных и условных вершин.
2. Каждый выход вершины ГСА соединяется только с одним входом.
3. Входы и выходы различных вершин соединяются дугами, направленными от выхода к входу.
4. Для любой вершины ГСА существует, по крайней мере, один путь из этой вершины к конечной вершине, проходящий через операторные и условные вершины в направлении соединяющих их дуг.
5. В каждой операторной вершине записываются микрооперации y_i, соответствующие одной микрокоманде У.
6. В каждой условной вершине записывается один из элементов множества логических условий x_i.
7. Начальной вершине ставится в соответствие фиктивный оператор y_0 а конечной — фиктивный оператор y_k. На рис. 3.4 показан пример микропрограммы, записанной на языке ГСА.

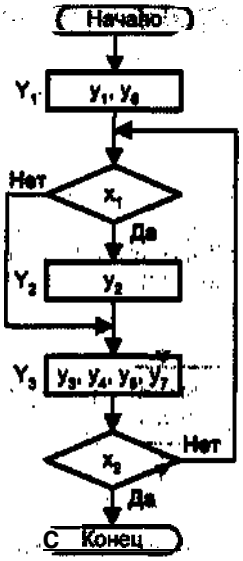


Рис. 3.4: Пример граф-схемы микропрограммы

В примере микрокоманда Y_1 инициирует микрооперации y_1 и y_6, микрокоманда Y_2 — микрооперацию y_2, а Y_3 микрооперации y_3, y_4, y_5 и y_7.

Языки микропрограммирования

Для детализированного задания микропрограмм используют языки микропрограммирования. *Языки микропрограммирования* (ЯМП) обеспечивают описание функционирования ВМ в терминах микроопераций.

Если средства языка ориентированы на запись микропрограммы без привязки к конкретным структурам для реализации этой микропрограммы, то такой ЯМП называют *языком функционального микропрограммирования,* а соответствующие микропрограммы - *функциональными микропрограммами* [21]. Функциональная микропрограмма используется как исходная форма для описания функционирования ВМ.

В случае когда средства языка нацелены на описание микропрограмм, привязанных к конкретной реализующей их структуре, ЯМП называют *языком структурно -функционального микропрограммирования.*

В последующих разделах для описания функционирования ВМ будет использоваться именно язык микропрограммирования, а конкретно вариант ЯМП, предложенный в [25]. Ниже рассматриваются основные средства языка.

Описание слов, шин, регистров

Основным элементом данных, с которым оперирует микропрограмма, является *слово.*

Описание слова состоит из названия (идентификатора) и разрядного указателя. Идентификатором может быть произвольная последовательность букв и цифр, начинающаяся с буквы. Разрядный указатель состоит из номеров старшего и младшего разрядов слова, разделенных горизонтальной чертой (дефис). Номер старшего разряда записывается слева от черты, а номер младшего — справа. Указатель заключается в круглые скобки. Так, описание слова, представляющего 32-разрядный адрес $_{\text{исп}} = a_{31}, a_{30}, \ldots, a_0$ записывается в виде $A_{\text{исп}}(31\text{-}0)$. Разрядный указатель может опускаться, если это не вызывает недоразумений (например, если слово уже было описано раньше).

В структуре вычислительной машины важную роль играют шины. Шиной называется совокупность цепей, используемых для передачи слов. Одна цепь обеспечивает передачу бита информации. Описание шины, как и слова, состоит из идентификатора и разрядного указателя. Например, описание 32-разрядной шины адреса имеет вид ША(31-0).

Описание регистра также включает в себя названия регистра и разрядного указателя. Приведем примеры. Так, пусть команда имеет длину 32 бита и состоит из 8-разрядного кода операции, 4-разрядного поля способа адресации и 20-разрядного поля адреса. Тогда описание регистра команды выглядит следующим образом: РК(31-0), а описания его отдельных элементов и соответственно полей команды имеют вид: РК(31-24), РК(23-2О), РК(19-О). Вместо номеров разрядов в разрядном указателе можно записывать наименование поля слова. Тогда два первых поля регистра команды могут быть представлены так: РК(КОП), РК(СА).

Описание 32-разрядного регистра РПЗ для хранения чисел с плавающей запятой, где число состоит из трех полей: s (поле знака мантиссы, бит 31), p (поле порядка, биты 30-23) и m (поле мантиссы, биты 22-0), задается в виде РПЗ(31 • 30--23 • 22–0)или РПЗ(s • p • m). Здесь точка обозначает операцию составления целого слова из его частей.

Описание памяти, слова памяти

В самом общем виде описание памяти емкостью 1000 16-разрядных слов имеет вид: ПАМ [000:999] (15-0). Здесь ПАМ — стандартное название памяти. Мы в дальнейшем будем использовать следующие идентификаторы памяти: ОП (основная память), ОЗУ (модуль оперативного запоминающего устройства), ПЗУ (модуль постоянного запоминающего устройства). В квадратных скобках записывается адресный указатель (слева от двоеточия адрес первого, а справа - адрес последнего слова памяти). Наконец, в круглые скобки заключается разрядный указатель слова (все слова памяти имеют одинаковую разрядность).

Примеры.

Описания модулей ОЗУ, содержащих по 1 Кбайт (1024 байта):

03У1 [0000:1023](7-0); 03У2[1024:2047](7-0).

Описания модулей ПЗУ, содержащих по 8192 32-разрядных слова:

ПЗУ1[000016:0FFF16](0-31),ПЗУ2[100016:lFFFl6](0-31).

Здесь адреса слов указаны в шестнадцатеричном коде, в каждом слове старший разряд имеет номер 0, а младший - 31.

Описание слова памяти поделено на две части: идентификатор области памяти и адресный указатель слова (в квадратных скобках). Допускается символическая запись адреса, а также косвенное указание адреса слова.

Примеры описаний слов памяти: 03У1[211], или 03У1 [Аисп], или 03У1[(РАП)], где Аисп - символический адрес, (РАП) - косвенный адрес, значение которого содержится в регистре РАП.

Описание микроопераций

Здесь под микрооперацией понимается элементарная функциональная операция, выполняемая над словами под воздействием одного сигнала управления, который вырабатывается устройством управления ВМ. В зависимости от количества преобразуемых слов (операндов) различают одноместные, двухместные и трехместные микрооперации.

Описание микрооперации складывается из двух частей, разделяемых двоеточием

МЕТКА: МИКРООПЕРАТОР.

Метка — это обозначение сигнала управления, вызывающего выполнение микрооперации. Метка принимает два значения: 1 - микрооперация выполняется, 0 - не выполняется.

Микрооператор определяет содержимое производимого элементарного действия (микрооперации).

Например, микрооператор записи в регистр С результата сложения слов из регистров А и В имеет вид:

РгС(15-О) :=РгА(15-0) + РгВ(15-0),

а полное описание микрооперации принимает форму

у15: **РгС(15–0) :** РгА(15-0) + РгВ(15-0).

Здесь указано, что микрооперация инициируется сигналом управления у15.

Для повышения наглядности записей желательно, чтобы метка сигнала управления несла смысловую нагрузку. Например, для микроопераций выдачи информации идентификатор метки сигнала можно начинать с буквы «В», а для приема — с буквы **«П»;** метку для микрооперации пересылки из регистра РА в регистр РВ можно записать в виде РАРВ.

Микрооператор по форме записи представляет собой оператор присваивания. Выражение справа от знака присваивания (:=) называется формулой микрооператора. Формула определяет вид **преобразования,** производимого микрооперацией, и местоположение преобразуемых операндов. Слева от знака присваивания в микрооператоре указывается приемник результата реализации формулы.

В соответствии с формулой микрооператора будем различать следующие классы микроопераций.

Микрооперация установки - присваивание слову значения константы.

Например, ПРгХ **РгХ(s** • m): 0; **ПРгС:С(7–0)** := 3110.

Микрооперации передачи — присваивание слову значения другого слова, в том числе с инверсией передаваемого слова.

Примером простого присваивания может служить микрооперация БпУп СК **:=**РА Здесь микрооператор описывает занесение в счетчик команд содержимого регистра адреса (адресного поля регистра команды), то есть реализацию перехода в командах безусловного и условного перехода, что и отражает идентификатор сигнала управления.

Другие примеры микроопераций пересылки:

РХРУ: РгУ(15–0) : РгХ(15-0); РевХУ: РгУ(15–0) : РгХ(0-15).

Первый микрооператор описывает пересылку 16-разрядного слова из регистра РгХ в регистр **РгУ** с сохранением расположения разрядов, а второй — с **«разворотом»** исходного слова.

Микрооперации передачи числа с плавающей запятой, имеющего поля знака s, порядка р и мантиссы **m,** а также передачи знака с инвертированием, имеют вид:

Пз1Пз2: РПЗ2(S • р • m) := РПз1 (s • р • m); ПИЗ: РгХ(s) := РгУ(s).

Если регистры связаны между собой не непосредственно, а через шину, которая используется многими **источниками** и приемниками данных, то передача слова между ними возможна при одновременном выполнении двух **микроопераций,** и описание принимает вид:

ВРгВ:ША:=РгВ; ПРгА: РгА:= ША

или

ПРгА, ВРгВ: РгА := ША : РгВ.

Здесь метки одновременно формируемых сигналов управления перечисляются через запятую и образуют микрокоманду.

Микрооперации составления слова — обеспечивают получение целого слова большой разрядности из нескольких малоразрядных слов.

Пусть в 16-разрядный регистр А нужно передать слово, старшие разряды которого содержатся в 8-разрядном регистре В, а младшие - в 8-разрядном регистре С. Соответствующую микрооперацию можно описать так:

ПРгА: PrA(15-0):= PrB(7-0) • PrC(7–0),

где точка (•) - знак присоединения.

Операция присоединения предназначена для присоединения значения слова, указанного справа от знака операции, к значению слова, расположенного слева от знака операции.

Микрооперации сдвига служат для изменения положения разрядов слова. Положение разрядов изменяется путем перемещения каждого разряда на несколько позиций влево или вправо.

Микрооперации сдвига слова в аккумуляторе, например, могут быть описаны в следующих формах:

- R 2 A K : AK(15-0):= PC(1-0) • AK(15-2) — сдвиг на два разряда вправо с введением в два старших освобождающихся разряда содержимого двух младших разрядов регистра PC;
- L 1AK AK(15-0):= AK(14-0) • 0 — сдвиг на один разряд влево с занесением в освобождающийся разряд нуля;
- **R2AK(15–0):** PC(15-0):= AK(15) • AK(15) • AK(15-2) - арифметический сдвиг слова вправо на два разряда с загрузкой в старшие освобождающиеся разряды знака.

Для сокращения записи микрооперации сдвига используются две процедуры:

- Rn(A) — удаление п младших правых разрядов из слова А, то есть сдвиг значения на п разрядов вправо;
- Ln(A) — удаление п старших левых разрядов из слова А, то есть сдвиг значения на п разрядов влево.

Использование этих процедур приводит к представлению ранее рассмотренных микрооператоров в форме:

- **AK(15-0):=R2(PC(1-0) • AK);**
- **AK(15-0):= L1(AK • 0);**
- **PC(15-0):=R2(AK(s)•AK(s)•AK).**

Микрооперация счета — обеспечивает изменение значения слова на единицу:

+1СК:СК:=СК+1.

Микрооперация сложения — служит для присваивания слову суммы слагаемых:

СлАд:РАП:=ИР + РА

Логические микрооперации — присваивают слову значение, полученное поразрядным применением функций И (^), ИЛИ (v), исключающее ИЛИ к парам соответствующих разрядов операндов:

И: АК :=PrX л PrY; М2: АК := PrX ⊕ PrY.

Микрооперация двоичного декодирования — состоит в преобразовании n-разрядного двоичного позиционного кода *A* в 2"-разрядный унитарный код *B*. В унитарном коде только один разряд принимает единичное значение, а все остальные равны нулю. Номер разряда *K*, который принимает значение 1, определяется значением кода $A = a_{n-1}, a_{n-2}, ..., a_0$:

$$K = \sum_{i=0}^{n} a_i \times 2^i.$$

Принято следующее условное обозначение: B := decod (A).

Комментарии к микрооперациям

Микрооперации могут снабжаться произвольными комментариями. Комментарии записываются справа от микрооперации и заключаются в угловые скобки. Например,

+1СК := СК := СК + 1 <Увеличение содержимого СК на единицу>

Совместимость микроопераций

Совместимостью называется свойство совокупности микроопераций, гарантирующее возможность их параллельного выполнения [21]. Различают функциональную и структурную совместимости. Пусть S_1, S_2, S_3, S_4 — подмножества слов из множества 5. Тогда микрооперации $S_1 := \varphi_1(S_2)$ и $S_3 := \varphi_2(S_4)$ называются *функционально совместимыми,* если $S_1 \cap S_2$ Ш, то есть если микрооперации присваивают значения разным словам. В функциональных микропрограммах, описывающих алгоритмы выполнения операций без учета структуры вычислительной машины, одновременно могут выполняться только функционально совместимые микрооперации.

Структура ВМ может внести дополнительные ограничения на возможность параллельного выполнения микроопераций. Микрооперации называются *структурно несовместимыми,* если из-за ограничений, обусловленных структурой ВМ, они не могут выполняться параллельно. Обычно структурная • несовместимость связана с использованием микрооперациями одного и того же оборудования.

Цикл команды

Программа в фон-неймановской ЭВМ реализуется центральным процессором (ЦП) посредством последовательного исполнения образующих эту программу команд. Действия, требуемые для выборки (извлечения из основной памяти) и выполнения команды, называют *циклом команды.* В общем случае цикл команды включает в себя несколько составляющих (этапов):

- выборку команды;
- формирование адреса следующей команды;
- декодирование команды;
- вычисление адресов операндов;
- выборку операндов;

- исполнение операции;
- запись результата.

Перечисленные этапы выполнения команды в дальнейшем будем называть *стандартным циклом команды*. Отметим, что не все из этапов присутствуют при выполнении любой команды (зависит от типа команды), тем не менее этапы выборки, декодирования, формирования адреса следующей команды и исполнения имеют место всегда.

В определенных ситуациях возможны еще два этапа:

- косвенная адресация;
- реакция на прерывание.

Стандартный цикл команды

Кратко охарактеризуем каждый из вышеперечисленных этапов стандартного цикла команды. При изучении данного материала следует учитывать, что приводимое описание имеет целью лишь дать представление о сущности каждого из этапов. В то же время распределение функций по разным этапам цикла команды и последовательность выполнения некоторых из них в реальных М могут отличаться от излагаемых.

Этап выборки команды

Цикл любой команды начинается с того, что центральный процессор извлекает команду из памяти, используя адрес, хранящийся в счетчике команд (СК). Двоичный код команды помещается в регистр команды (РК) и с этого момента становится «видимым» для процессора. Без учета промежуточных пересылок и сигналов управления это можно описать следующим образом: РК := ОП [(СК)].

Приведенная запись охватывает весь этап выборки, если длина команды совпадает с разрядностью ячейки памяти. В то же время система команд многих ВМ предполагает несколько форматов команд, причем в разных форматах команда может занимать , 2 или более ячеек, а этап выборки команды можно считать завершенным лишь после того, как в РК будет помещен полный код команды. Информация о фактической длине команды содержится в полях кода операции и способа адресации. Обычно эти поля располагают в первом слове кода команды, и для выяснения необходимости продолжения процесса выборки необходимо предварительное декодирование их содержимого. Такое декодирование может быть произведено после того, как первое слово кода команды окажется в РК. В случае многословного формата команды процесс выборки продолжается вплоть до занесения в РК всех слов команды. Например, для 16-разрядной команды, занимающей две 8-разрядные ячейки памяти, выборку можно описать так:

```
ПСтРК:РК(15-8):=0П[(СК)];
+1СК:СК:=СК + 1;
ПМлРК:РК(7-0):=ОП[(СК)].
```

Этап формирования адреса следующей команды

Для фон-неймановских машин характерно размещение соседних команд программы в смежных ячейках памяти. Если извлеченная команда не нарушает естествен-

ного порядка выполнения программы, для вычисления адреса следующей выполняемой команды достаточно увеличить содержимое счетчика команд на длину текущей команды, представленную количеством занимаемых кодом команды ячеек памяти. Для однословной команды это описывается микрооперацией: +1СК: СК := СК + .

Длина команды, а также то, способна ли она изменить естественный порядок выполнения команд программы, выясняются в ходе ранее упоминавшегося предварительного декодирования. Если извлеченная команда способна изменить последовательность выполнения программы (команда условного или безусловного перехода, вызова процедуры и т. п.), процесс формирования адреса следующей команды переносится на этап исполнения операции. В силу сказанного, в ряде ВМ рассматриваемый этап цикла команды следует не за выборкой команды, а находится в конце цикла.

Этап декодирования команды

После выборки команды она должна быть декодирована, для чего ЦП расшифровывает находящийся в РК код команды. В результате декодирования выясняются следующие моменты:

- находится ли в РК полный код команды или требуется дозагрузка остальных слов команды;
- какие последующие действия нужны для выполнения данной команды;
- если команда использует операнды, то откуда они должны быть взяты (номер регистра или адрес ячейки основной памяти);
- если команда формирует результат,о куда этот результат должен быть направлен.

Ответы на два первых вопроса дает расшифровка кода операции, результатом которой может быть унитарный код, где каждый разряд соответствует одной из команд, что можно описать в виде УнитК:=decod(Коп). На практике вместо унитарного кода могут встретиться самые разнообразные формы представления результатов декодирования, например адрес ячейки специальной управляющей памяти, где хранится первая микрокоманда микропрограммы для реализации указанной в команде операции.

Полное выяснение всех аспектов команды, помимо расшифровки кода операции, требует также анализа адресной части команды, включая поле способа адресации.

По результатам декодирования производится подготовка электронных схем ВМ к выполнению предписанных командой действий.

Этап вычисления адресов операндов

Этап имеет место, если в процессе декодирования команды выясняется, что команда использует операнды. Если операнды размещаются в основной памяти, осуществляется. вычисление их исполнительных адресов, с учетом указанного в команде способа адресации. *Так,* в случае индексной адресации для получения исполнительного адреса производится суммирование содержимого адресной части команды и содержимого индексного регистра.

Этап выборки операндов

Вычисленные на предыдущем этапе исполнительные адреса используются для считывания операндов из памяти и занесения в определенные регистры процессора. Например, в случае арифметической команды операнд после извлечения из памяти может быть загружен во входной регистр АЛУ. Однако чаще операнды предварительно заносятся в специальные вспомогательные регистры процессора, а их пересылка на вход АЛУ происходит на этапе исполнения операции.

Этап исполнения операции

На этом этапе реализуется указанная в команде операция. В силу различия сущности каждой из команд ВМ, содержание этого этапа также сугубо индивидуально. Этапы исполнения некоторых команд будут рассмотрены ниже на примере выполнения учебной программы для приведенной на рис. 3.1 гипотетической вычислительной машины.

Этап записи результата

Этап записи результата присутствует в цикле тех команд, которые предполагают занесение результата в регистр или ячейку основной памяти. Фактически его можно считать частью этапа исполнения, особенно для тех команд, которые помещают результат сразу в несколько мест.

Описание стандартных циклов команды для гипотетической машины

Для анализа содержания стандартных циклов команды обратимся к гипотетической ВМ (см. рис. 3.1), для которой составим программу сложения двух чисел и вывода суммы на устройство вывода, если эта сумма не равна 0.

Список необходимых команд приведен в табл. 3.1, а сама программа - в табл. 3.2. Предполагается, что команды программы будут размещаться в основной памяти, начиная с адреса 100, а операнды — с адреса 200. Для вывода результата назначим порт с номером 5.

Таблица 3.1. Команды гипотетической вычислительной машины

КОп	АЧ	Описание
LDA	Adr	Загрузка в аккумулятор содержимого ячейки ОП с адресом Adr
ADD	Adr	Сложение содержимого аккумулятора с содержимым ячейки ОП, имеющей адрес Adr. Результат остается в аккумуляторе
BRZ	Adr	Переход к команде, хранящейся по адресу Adr, если результат предыдущей арифметической операции равен 0, иначе естественный порядок вычислений не нарушается
OUT	PortN	Вывод содержимого аккумулятора на периферийное устройство, подключенное к порту с номером PortN
HLT		Останов вычислений

Таблица 3.2. Программа к рассматриваемому примеру

Адрес порта	Код операции	Адрес ячейки или номер
100	LDA	200
101	ADD	201
102	BRZ	104
103	OUT	5
104	HLT	
200	Операнд 1	
201	Операнд 2	

Перед запуском программы необходимо занести в СК адрес ячейки основной памяти, содержащей первую выполняемую команду программы, то есть 100.

Поскольку выборка и декодирование, а также формирование адреса следующей команды для всех команд выполняются по идентичной схеме, опишем их однократно, детализируя в дальнейшем лишь остальные этапы основного цикла команды. Кроме того, напомним, что все сигналы управления и управляющие коды формируются микропрограммным автоматом, поэтому в дальнейшем ссылки на МПА будут опущены.

Выборка команды. Сначала остановимся на содержании этапа выборки, идентичного для всех команд программы. На этом этапе происходит извлечение двоичного кода команды из ячейки основной памяти и его занесение в регистр команды:

СКРАП: РАП := СК, ЧтЗУ: РД1 := ОП[(СК)];
РДПКОп, РДПРА: РК :=РДП <РКОп := РДП(КОп), РА := РДП(АЧ)>.

В первом такте вырабатывается сигнал управления СКРАП, инициирующий пересылку содержимого счетчика команд в регистр адреса памяти. По сигналу ЧтЗУ содержимое ячейки, выбранной дешифратором адреса памяти (код команды), переписывается в регистр данных памяти. В следующем такте формируются сигналы РДПКОп и РДПРА , по которым содержимое РДП передается в РК, при этом поле РКОп заполняется кодом операции, а поле РА — адресной частью команды.

Декодирование команды. Сразу же после размещения кода операции в РК производится его декодирование, что можно описать микрооператором УнитК := decod(Коп). Прежде всего выясняется, может ли данная команда изменить последовательность вычислений, что влияет на дальнейшее выполнение цикла команды. Для всех команд, кроме команд управления (в нашем случае это BRZ), начинается этап формирования адреса следующей команды.

Формирование адреса следующей команды. В фон-неймановских ВМ команды программы располагаются в естественном порядке следования, в соседних ячейках памяти, и выполняются в том же порядке. Для формирования адреса следующей команды достаточно увеличить содержимое СК на единицу: +1СК СК := СК + 1.

В рассматриваемой ВМ предусмотрена только прямая адресация, поэтому этап вычисления адресов операндов опускается.

В анализируемой программе этап выборки операндов предполагается только в командах загрузки аккумулятора и сложения. Для простоты изложения будем расценивать выборку операндов как часть этапа исполнения соответствующих операций. В свою очередь, этапы исполнения специфичны для каждой команды и рассматриваются применительно к каждой команде нашей программы.

Исполнение операции загрузки аккумулятора. Команда LDA 200 обеспечивает занесение в аккумулятор содержимого ячейки ОП с адресом 200, то есть первого операнда, и реализуется следующим образом:

- МПА вырабатывает сигнал РАРАП, передающий содержимое РА (адресную часть команды) в РАП;
- по сигналу ЧтЗУ содержимое ячейки 200 заносится в РДП;
- по сигналу РДПАкк первый операнд из РДП помещается в аккумулятор.

На языке микроопераций это выглядит так:

РАРАП: РАП := РК(РА); ЧтЗУ: РДП := ОП[200];
РДПАкк: Акк := РДП.

Исполнение операции сложения. Команда ADD 201 обеспечивает суммирование текущего содержимого аккумулятора с содержимым ячейки 201 (вторым операндом). Результат сложения остается в аккумуляторе. Одновременно с этим в АЛУ формируются признаки результата:

- МПА вырабатывает сигнал РАРАП, и содержимое РА поступает в РАП;
- по сигналу ЧтЗУ содержимое ячейки 201 заносится в РДП;
- сигнал управления РДП РХ вызывает пересылку операнда 2 из РДП в регистр РХ АЛУ одновременно с этим МПА вырабатывает сигнал АККРУ, ПО которому в РУ переписывается содержимое аккумулятора, то есть хранящийся там первый операнд; операционный блок выполняет над данными, расположенными в РХ и РУ, операцию, заданную в коде операции команды (в нашем случае — сложение);
- по сигналу ОПБАкк информация с выхода ОПБ загружается в аккумулятор.

Сказанное может быть описано в виде:

РАРАП РАП := РК(РА), ЧтЗУ: РДП := ОП[201];
РДПРХ: РХ := РДП, АККРУ: РУ := Акк, ОПБ := РХ + РУ;
ОПБАкк1: Акк:-ОПБ.

Исполнение операции условного перехода. Для изменения порядка выполнения программы используются команды безусловного (БП) и условного (УП) переходов, в нашем случае — команда BRZ 104. Адрес перехода хранится в адресном поле команды. Команда анализирует хранящийся в РПрз признак (флаг) нулевого результата, выработанный в АЛУ на предыдущем этапе вычислений, и формирует адрес следующей команды в зависимости от состояния этого признака. При нулевом значении флага (условие перехода не выполнено) естественный порядок выполнения программы не нарушается, и адрес следующей команды формируется обычным образом, путем увеличения содержимого СК на единицу (+1GK СК := СК +1). При единичном значении флага (условие перехода выполнено) в СК зано-

сится содержимое РА. Напомним, что в РА находится адресная часть извлеченной из ОП команды перехода, то есть адрес точки перехода. Сказанное можно записать в виде БПУП СК=РК(РА). Поскольку'для команд перехода формирование адреса следующей команды по сути является и исполнением их операций, описанные микрооперации для них обычно относят к этапу исполнения.

Исполнение операции вывода. Команда OUT 5 обеспечивает вывод содержимого аккумулятора на периферийное устройство (ПУ), подключенное к порту вывода с номером 5. МП А вырабатывает управляющий сигнал РАДВВ, по которому адресная часть команды — номер порта вывода — из РА поступает на вход дешифратора номера порта ввода/вывода. В следующем такте по сигналу Вывсодержимое аккумулятора через выбранный дешифратором порт вывода передается на подключенное к этому порту ПУ:

РАДВВ: ДВВ:=РК(РА);
Вьв: Порт вьвода 5 := Акк.

Исполнение операции останова. Команда HLT приводит к завершению вычислений. При этом вырабатывается сигнал Ост, нужный для того, чтобы известить операционную систему о завершении текущей программы.

Машинный цикл с косвенной адресацией

Многие команды предполагают чтение операндов из памяти или запись в память. В простейшем случае в адресном поле таких команд явно указывается исполнительный адрес соответствующей ячейки ОП. Однако часто используется и другой способ указания адреса, когда адрес операнда хранится в какой-то ячейке памяти, а в команде указывается адрес,ячейки, содержащей адрес операнда. Как уже отмечалось ранее, подобный прием называется *косвенной адресацией*. Чтобы прочитать или записать операнд, сначала нужно извлечь из памяти его адрес и только после этого произвести нужное действие (чтение или запись операнда), иными словами, требуется выполнить два обращения к памяти. Это, естественно, отражается и на цикле команды, в котором появляется косвенная адресация. Этап косвенной адресации можно отнести к этапу вычисления адресов операндов, поскольку его сущность сводится к определению исполнительного адреса операнда.

Применительно к вычислительной машине, приведенной на рис. 3.1, при косвенной адресации имеют место следующие микрооперации:

РАРАП РАП := РК(РА), ЧтЗУ: РДП:= ОП[(РА)];
РДПРА: РК(РА):= РДП.

Иными словами, содержимое адресного поля команды в регистре команд используется для обращения к ячейке ОП, в которой хранится адрес операнда, после чего извлеченный из памяти исполнительный адрес операнда помещается в адресное поле регистра команды на место косвенного адреса. Дальнейшее выполнение команды протекает стандартным образом.

Машинный цикл с прерыванием

Практически во всех ВМ предусмотрены средства, благодаря которым модули ввода/вывода (и не только они) могут прервать выполнение текущей программы для

внеочередного выполнения другой программы, с последующим возвратом к прерванной.

Первоначально прерывания были введены для повышения эффективности вычислений при работе с медленными ПУ. Положим, что процессор пересылает данные на принтер, используя стандартный цикл команды. После каждой операции записи ЦП будет вынужден сделать паузу, в ожидании подтверждения от принтера об обработке символа. Длительность этой паузы может составлять сотни и тысячи циклов команды. Ясно, что такое использование ЦП очень неэффективно. В случае прерываний, пока протекает операция ввода/вывода, ЦП способен выполнять другие команды.

В упрощенном виде процедуру прерывания можно описать следующим образом. Объект, требующий внеочередного обслуживания, выставляет на соответствующем входе ЦП сигнал *запроса прерывания.* Перед переходом к очередному циклу команды процессор проверяет этот вход на наличие запроса. Обнаружив запрос, ЦП запоминает информацию, необходимую для продолжения нормальной работы после возврата из прерывания, и переходит к выполнению *программы обработки прерывания* (обработчика прерывания). По завершении обработки прерывания ЦП восстанавливает состояние прерванного процесса, используя запомненную информацию, и продолжает вычисления. Описанный процесс иллюстрирует рис. 3.5.

В терминах цикла команды сказанное выглядит так. Для учета прерываний к циклу команды добавляется этап прерывания, в ходе которого процессор проверяет, не поступил ли запрос прерывания. Если запроса нет, ЦП переходит к этапу выборки следующей команды программы. При наличии запроса процессор:

1. Приостанавливает выполнение текущей программы и запоминает содержимое всех регистров, которые будут использоваться программой обработки прерывания. Это называется *сохранением контекста программы.* В первую очередь необходимо сохранить содержимое счетчика команд, аккумулятора и регистра признаков. Контекст программы обычно сохраняется в стеке.
2. Заносит в счетчик команд начальный адрес программы обработки прерывания.

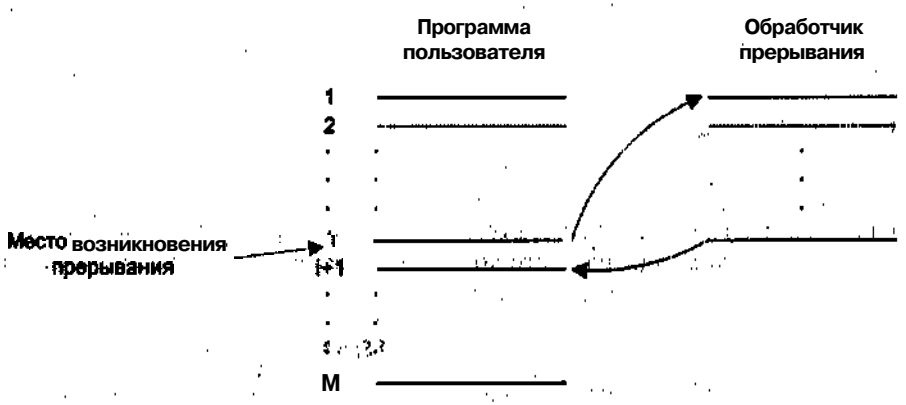


Рис. 3.5. Передача управления при прерываниях

На языке микроопераций это можно записать следующим образом:

СКРДП: РДП := СК; -1УС: УС := УС -1 «Запись в стек содержимого СК>;
УСРАП: РАП := УС; ЗпЗУ, СКРДП: ОП[(УС)] := РДП :=СК;
АккРДП: РДП := Акк; -1УС: УС := УС - 1 «Запись в стек содержимого Акк>;
УСРАП: РАП := УС; ЗпЗУ, АккРДП: ОП[(УС)] := РДП := Акк;
РПрзРДП: РДП := РПрз; -1УС: УС := УС -1 «Запись в стек содержимого РПрз>;
УСРАП: РАП := УС; ЗпЗУ, РПрзРДП: ОП[(УС)] := РДП := РПрз;
СК:= Аобр «Занесение в СК начального адреса обработчика прерываниям

Теперь процессор продолжает с этапа выборки первой команды обработчика прерывания. Обработчик (обычно он входит в состав операционной системы) определяет природу прерывания и выполняет необходимые действия. Когда программа обработки прерывания завершается, процессор может возобновить выполнение программы пользователя с точки, где она была прервана. Для этого он восстанавливает контекст программы (содержимое СК и других регистров) и начинает с цикла выборки очередной команды прерванной программы. Соответствующая микропрограмма представлена ниже (с целью упрощения схемы некоторые тракты и сигналы управления не показаны, и соответствующие микрооперации отображают лишь логику выхода из прерывания).

УСРАП: РАП :=УС, ЧтЗУ: РДП:=ОП[(УС)] «Восстановление содержимого РПрз>;
+1УС: УС:- УС +1; РДПРПрз: РПрз := РДП;
УСРАП: РАП :=УС, ЧгЗУ: РДП:=ОП[(УС)] восстановление содержимого Акк>;
+1УС: УС:= УС + 1; РДПАкк: Акк:= РДП;
УСРАП: РАП := УС, ЧтЗУ: РДП:=ОП[(УС)] «Восстановление содержимого СК>;
+1УС: УС :=УС + 1; РДПСК: СК:=РДП.

Диаграмма состояний цикла команды

Вышеизложенное можно подытожить в виде рис. 3.6, где показаны потоки информации в ходе этапов выборки команды, косвенной адресации и прерывания [36]:

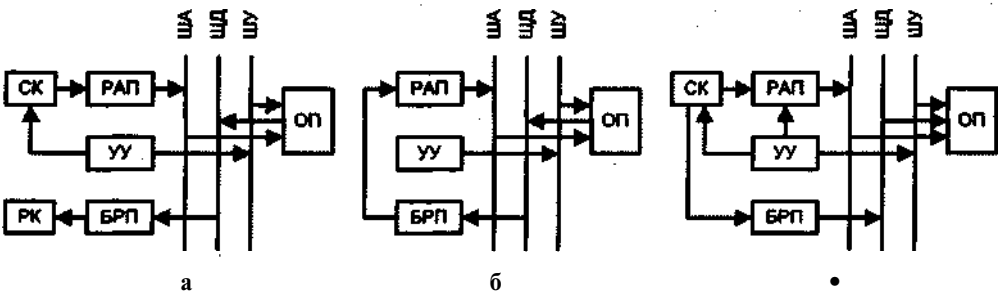


Рис. 3.6. Потоки информации при реализации цикла команды: а - этап выборки; б - этап косвенной адресации; *в* — этап прерывания .

В работе [36] использован также иной подход к описанию содержания цикла команды — с помощью диаграммы состояний (рис. 3.7).

На такой диаграмме цикл команды представляется в виде последовательности состояний. Для каждой конкретной команды некоторые состояния могут быть

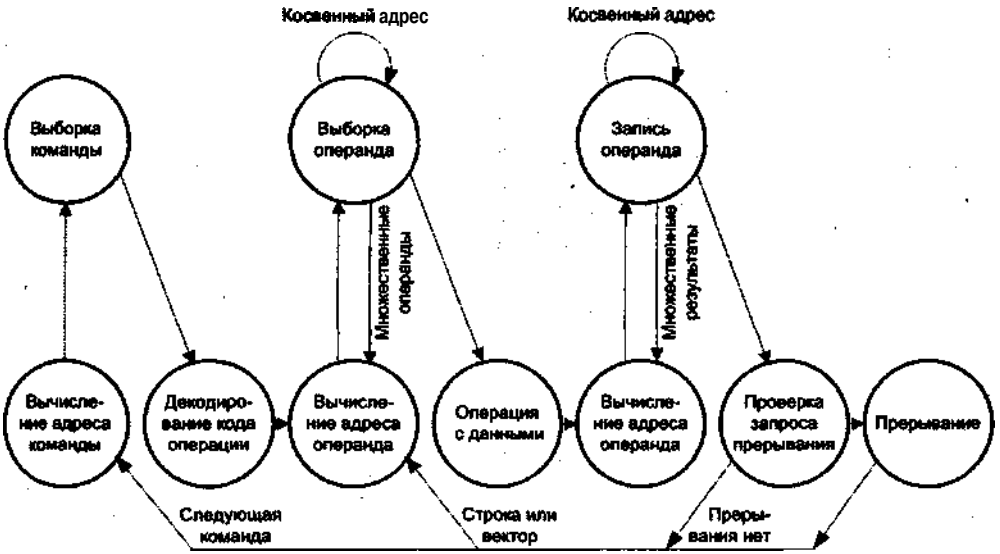


Рис. 3.7. Диаграмма состояний цикла команды

нулевыми, а некоторые другие могут неоднократно повторяться. Полный цикл команды может включать в себя следующие состояния:

- **Вычисление адреса команды.** Определение исполнительного адреса команды, которая должна выполняться следующей.
- **Выборка команды.** Чтение команды из ячейки памяти и занесение ее в РК.
- **Декодирование команды.** Анализ команды с целью выяснения типа подлежащей выполнению операции и операндов.
- **Вычисление адреса операнда.** Определение исполнительного адреса операнда, если операция предполагает обращение к операнду, хранящемуся в памяти или же доступному посредством ввода.
- **Выборка операнда.** Выборка операнда из памяти или его ввод с устройства ввода.
- **Операция с данными.** Выполнение операции, указанной в команде.
- **Запись операнда.** Запись результата в память или вывод на устройство вывода.

Состояния в верхней части диаграммы описывают обмен. между ЦП и памятью либо между ЦП и модулем ввода/вывода. Состояния в нижней части обозначают только внутренние операции ЦП. Вычисление адреса операнда встречается дважды, поскольку команда может включать в себя чтение, запись или и то и другое, однако действия, выполняемые в этом состоянии, в обоих случаях одни и те же, поэтому используется один и тот же идентификатор состояния.

Следует отметить, что диаграмма допускает множественные операнды и результаты, как того требуют некоторые команды. Кроме того, в ряде ВМ единственная команда может определять операцию над вектором (одномерным массивом чисел)

или строкой (одномерным массивом символов), что требует повторяющихся операций выборки и/или записи. Диаграмма отражает также возможность этапов прерывания и косвенной адресации.

Основные показатели вычислительных машин

Использование конкретной вычислительной машины имеет смысл, если ее показатели соответствуют показателям, определяемым требованиями к реализации заданных алгоритмов. В качестве основных показателей ВМ обычно рассматривают: емкость памяти, быстродействие и производительность, стоимость и надежность [25]. В данном учебнике остановимся только на показателях быстродействия и производительности, обычно представляющих основной интерес для пользователей.

Быстродействие

Целесообразно рассматривать два вида быстродействия: номинальное и среднее.

Номинальное быстродействие характеризует возможности ВМ при выполнении стандартной операции. В качестве стандартной обычно выбирают короткую операцию сложения. Если обозначить через $\tau_{сл}$ время сложения, то номинальное быстродействие определится из выражения

$$V_{ном} = \frac{1}{\tau_{сл}} \left[\frac{оп}{с}\right].$$

Среднее быстродействие характеризует скорость вычислений при выполнени эталонного алгоритма или некоторого класса алгоритмов. Величина среднего быстродействия зависит как от параметров ВМ, так и от параметров алгоритма и определяется соотношением

$$v_{ср} = \frac{N}{T_э},$$

где $T_э$ — время выполнения эталонного алгоритма; N — количество операций, содержащихся в эталонном алгоритме.

Обозначим через n_i число операций i-го типа; l - количество типов операций в алгоритме ($i = 1, 2, ..., l$); τ_i — время выполнения операции i-го типа.

Время выполнения эталонного алгоритма рассчитывается по формуле:

$$T_э = \sum_{i=1}^{l} \tau_i n_i. \qquad (3.1)$$

Подставив (3.1) в выражение для $V_{ср}$, получим

$$V_{ср} = \frac{N}{\sum_{i=1}^{l} \tau_i n_i}. \qquad (3.2)$$

Разделим числитель и знаменатель в (3.2) на N:

$$V_{\text{ср}} = \frac{1}{\sum_{i=1}^{l} \frac{n_i}{N} \tau_i} \qquad (3.3)$$

Обозначив частоту появления операции i-го типа в (3.3) через $q_i = \frac{n_i}{N}$, запишем окончательную формулу для расчета среднего быстродействия:

$$V_{\text{ср}} = \frac{1}{\sum_{i=1}^{l} q_i \tau_i} \left[\frac{\text{оп}}{\text{с}}\right]. \qquad (3.4)$$

В выражении (3.4) вектор $\{\tau_1, \tau_2, ..., \tau_l\}$ характеризует систему команд ВМ, а вектор $\{q_1, q_2, ..., q_l\}$, называемый частотным вектором операций, характеризует алгоритм.

Очевидно, что для эффективной реализации алгоритма необходимо стремиться к увеличению $V_{\text{ср}}$. Если $V_{\text{ном}}$ главным образом отталкивается от быстродействия элементной базы, то $V_{\text{ср}}$ очень сильно зависит от оптимальности выбора команд ВМ.

Формула (3.4) позволяет определить среднее быстродействие машины при реализации одного алгоритма. Рассмотрим более общий случай, когда полный алгоритм состоит из нескольких частных, периодически повторяемых алгоритмов. Среднее быстродействие при решении полной задачи рассчитывается по формуле:

$$V_{\text{ср}}^{n} = \frac{1}{\sum_{j=1}^{m} \sum_{i=1}^{l} \beta_j q_{ji} \tau_i} \left[\frac{\text{оп}}{\text{с}}\right], \qquad (3.5)$$

где m — количество частных алгоритмов; p_j — частота появления операций j-го частного алгоритма в полном алгоритме; q_{ij} — частота операций i-го типа в j-м частном алгоритме.

Обозначим через N_j и 7} - количество операций и период повторения j-го частного алгоритма; $T_{\max} = \max(T_1, ..., T_j, ..., T_m)$—период повторения полного алгоритма;

$a_j = T_{\max} / T_j$ цикличность включения j-ого частного алгоритма в полном алгоритме.

Тогда за время $T_{\max}$ в ВМ будет выполнено $N_{\max} = \sum_{j=1}^{m} \alpha_j N_j$ операций, а частоту появления операций j-го частного алгоритма в полном алгоритме можно определить из выражения

$$\beta_j = \frac{\alpha_j N_j}{N_{\max}}. \qquad (3.6)$$

Для расчета по формулам (3.5, 3.6) необходимо знать параметры ВМ, представленные вектором $\{\tau_1, \tau_2, \ldots, X/\}$, параметры каждого j-го частного алгоритма — вектор $\{q_{j1} q_{j2} \ldots q_{jl}\}$ и параметры полного алгоритма — вектор $\{\beta_1, \beta_2, \ldots, \beta_m\}$.

Производительность ВМ оценивается количеством эталонных алгоритмов, выполняемых в единицу времени:

$$П = \frac{1}{T_э} \left[\frac{\text{задач}}{\text{с}}\right] \text{J*}$$

Производительность при выполнении полного алгоритма оценивается по формуле:

$$П_п = \frac{1}{\sum_{j=1}^{m} \sum_{i=1}^{l} \alpha_j N_j q_{ij} \tau_i} \left[\frac{\text{задач}}{\text{с}}\right]. \qquad (3.7)$$

Производительность является более универсальным показателем, чем среднее быстродействие, поскольку в явном виде зависит от порядка прохождения задач через ВМ.

Критерии эффективности вычислительных машин

Вычислительную машину можно определить множеством показателей, характеризующих отдельные ее свойства. Возникает задача введения меры для оценки степени приспособленности ВМ к выполнению возложенных на нее функций - меры эффективности.

Эффективность определяет степень соответствия ВМ своему назначению. Она измеряется либо количеством затрат, необходимых для получения определенного результата, либо результатом, полученным при определенных затратах. Произвести сравнительный анализ эффективности нескольких ВМ, принять решение на использование конкретной машины позволяет критерий эффективности.

Критерий эффективности - это правило, служащее для сравнительной оценки качества вариантов ВМ. Критерий эффективности можно назвать правилом предпочтения сравниваемых вариантов.

Строятся критерии эффективности на основе частных показателей эффективности (показателей качества). Способ связи между частными показателями определяет вид критерия эффективности.

Способы построения критериев эффективности

Возможны следующие способы построения критериев из частных показателей.

Выделение главного показателя. Из совокупности частных показателей $A_1, A_2, \ldots, A_n$ выделяется один, например A_1, который принимается за главный. На остальные показатели накладываются ограничения:

$$A_i = A_{i\text{доп}} \ (i = 2, 3, \ldots, n),$$

где $A_{i\text{доп}}$ — допустимое значение i-го показателя. Например, если в качестве A_1 выбирается производительность, а на показатели надежности P и стоимо-

сти S накладываются ограничения, то критерий эффективности ВМ принимает вид:

$$П_{упр} \rightarrow \max, \quad P = P_{доп}, SS_{доп}.$$

Способ последовательных уступок. Все частные показатели нумеруются в порядке их важности: наиболее существенным считается показатель Л,, а наименее важным — A_n. Находится минимальное значение показателя $A_1 — min A_1$, (если нужно найти максимум, то достаточно изменить знак показателя).Затем делается «уступка» первому показателю дельтаA_1, и получается ограничение $min A,$ + ДЛ,.

На втором шаге отыскивается min_2 при ограничении , $< min A,$ + ДЛ,. После этого выбирается «уступка» для $_2$: $min A_2 + \Delta A_2$. На третьем шаге отыскивается $min A_3$ при ограничениях А, $< min A_1$ + АЛ,; $A_2 < min A_2 + \Delta A_2$ и т. д. На последнем шаге ищут $min A_n$ при ограничениях

$A_1 \le min A_1$+ ДЛ,;

$A_2 \le min A_2 + \Delta A_2$;

...

$A_{n-1} \le min A_{n-1} + \Delta A_{n-1}$;

Полученный на этом шаге вариант вычислительной машины и значения ее показателей $A_1 A_2$,., Л„ считаются окончательными. Недостатком данного способа (критерия) является неоднозначность выбора ДЛ,.

Отношение частных показателей. В этом случае критерий эффективности получают в виде:

$$\bar{Л}, = \frac{A_1, A_2, ..., A_n}{B_1, B_2, ..., B_m} \rightarrow max \qquad (3.8)$$

или в виде:

$$K_2 = \frac{B_1, B_2 \ldots B_m}{A_1, A_2 \ldots A_n} \rightarrow min, \qquad (3.9)$$

или в виде:

где A_i $(i = 1, 2, ..., n)$ - частные показатели, для которых желательно увеличение численных значений, а B_i $(i = 1, 2,..., m)$ — частные показатели, численные значения которых нужно уменьшить. В частном случае критерий может быть представлен в виде:

$$K_3 = \frac{B_1}{A_1} \rightarrow min. \qquad (3.10)$$

Наиболее популярной формой выражения (3.10) является критерий цены эффективного быстродействия

$$K_4 = \frac{S}{V_{ср}} \rightarrow min, \qquad (3.11)$$

где S - стоимость, $V_{ср}$ - среднее быстродействие ВМ. Формула критерия K_4 характеризует аппаратные затраты, приходящиеся на единицу быстродействия.

Аддитивная форма. Критерий эффективности имеет вид:

$$K_5 = \sum_{i=1}^{n} \alpha_i A_i \rightarrow max, \qquad (3.12)$$

где а,, α_2, ..., α_n — положительные и отрицательные весовые коэффициенты частных показателей. Положительные коэффициенты ставятся при тех показателях, которые желательно максимизировать, а отрицательные — при тех, которые желательно минимизировать.

Весовые коэффициенты могут быть определены методом экспертных оценок. Обычно они удовлетворяют условиям

$$0 \le \alpha_i < 1, \sum_{i=1}^{n} \alpha_i = 1. \qquad (3.13)$$

Основной недостаток критерия заключается в возможности взаимной компенсации частных показателей.

Мультипликативная форма. Критерий эффективности имеет вид

$$K_6 = \prod_{i=1}^{n} A_i^{\alpha_i} \to max, \qquad (3.14)$$

где, в частном случае, коэффициенты α_i полагают равными единице.

От мультипликативной формы можно перейти к аддитивной, используя выражение:

$$\lg K_6 = \sum_{i=1}^{n} \alpha_i \lg A_i. \qquad (3.15)$$

Критерий K_6 имеет тот же недостаток, что и критерий K_5.

Максиминная форма. Критерий эффективности описывается выражением:

$$K_7 = \min_i A_i \to max. \qquad (3.16)$$

Здесь реализована идея равномерного повышения уровня всех показателей за счет максимального «подтягивания» наихудшего из показателей (имеющего минимальное значение).

У максиминного критерия нет того недостатка, который присущ мультипликативному и аддитивному критериям.

Нормализация частных показателей

Частные показатели качества обычно имеют различную физическую природу и различные масштабы измерений, из-за чего их простое сравнение становится практически невозможным. Поэтому появляется задача приведения частных показателей к единому масштабу измерений, то есть их нормализация.

Рассмотрим отдельные способы нормализации.

Использование отклонения частного показателя от максимального.

$$\Delta A_i = A_{max_i} - A_i. \qquad (3.17)$$

В данном случае переходят к отклонениям показателей, однако способ не устраняет различия масштабов отклонений.

Использование безразмерной величины $\overline{A_i}$.

$$\overline{A_i} = \frac{A_{max_i} - A_i}{A_{max_i}}, \qquad (3.18)$$

$$\overline{A_i} = \frac{A_i}{A_{max_i}}. \qquad (3.19)$$

Формула (3.18) применяется тогда, когда уменьшение A_i приводит к увеличению (улучшению) значения аддитивной формулы критерия. Выражение (3.19) используется, когда к увеличению значения аддитивной формулы критерия приводит увеличение Д.

Учет приоритета частных показателей

Необходимость в учете приоритетов возникает в случае, когда частные показатели имеют различную степень важности.

Приоритет частных показателей задается с помощью ряда приоритета I, вектора приоритета $(b_1, ..., b_q, ..., b_n)$ и вектора весовых коэффициентов $(\alpha_1, \alpha_2, ..., \alpha_n)$.

Ряд приоритета представляет собой упорядоченное множество индексов частных показателей $I = (1,2,..., n)$. Он отражает чисто качественные отношения доминирования показателей, а именно отношения следующего типа: показатель Л, важнее показателя A_2, а показатель A_2 важнее показателя A_3 и т. д.

Элемент b_q вектора приоритета показывает, во сколько раз показатель A_q важнее показателя A_{q+1} (здесь A_q — показатель, которому отведен номер q в ряду приоритета). Если A_q и A_{q+1} имеют одинаковый ранг, то b_q - 1. Для удобства принимают

Компоненты векторов приоритета и весовых коэффициентов связаны между собой следующим отношением

$$b_q = \frac{\alpha_q}{\alpha_{q+1}}.$$

Зависимость, позволяющая по известным значениям b_i определить величину α_q, имеет вид:

$$\alpha_q = \frac{\prod_{i=q}^{n} b_i}{\sum_{q=1}^{n} \prod_{i=q}^{n} b_i}. \qquad (3.20)$$

Знание весовых коэффициентов позволяет учесть приоритет частных показателей.

Контрольные вопросы

1. Какую функцию выполняет счетчик команд и какой должна быть его разрядность?
2. Какое из полей регистра команд должно быть заполнено в первую очередь?
3. Какой адрес должен быть занесен в указатель стека при его инициализации?
4. Какими средствами компенсируется различие в быстродействии процессора и основной памяти?

5. На основании какой информации микропрограммный автомат формирует сигналы управления?
6. Можно ли считать наличие регистров операндов обязательным условием работы любого операционного блока?
7. Каким образом используется информация, хранящаяся в регистре признаков?
8. С каким понятием можно ассоциировать сигнал управления?
9. В чем состоит различие между микрокомандой и микрооперацией?
10. Какие существуют способы записи микропрограмм?
11. Перечислите основные правила составления граф-схемы алгоритма.
12. Как в предложенном языке микропрограммирования описывается разрядность шины?
13. Какие варианты описания слова памяти допускает язык микропрограммирования?
14. Описание каких видов микроопераций допускает рассмотренный в книге язык микропрограммирования?
15. Что подразумевает понятие «совместимость микроопераций»?
16. Какие из этапов цикла команды являются обязательными для всех команд?
17. Какие узлы ВМ участвуют в реализации этапа выборки команды?
18. Местоположение какого из этапов цикла команды в общей их последовательности в принципе может быть изменено?
19. На какой стадии выполнения команды анализируются запросы прерывания?
20. Опишите последовательность действий, выполняемых при поступлении запроса прерывания.
21. Как обеспечивается возобновление вычислений после обработки прерывания?
22. Что понимается под номинальным и средним быстродействием ВМ?
23. Каким образом можно охарактеризовать производительность вычислительной машины?
24. Перечислите и охарактеризуйте основные способы построения критериев эффективности ВМ.
25. Какими способами можно произвести нормализацию частных показателей эффективности? ....

Глава 4
Организация шин

Совокупность трактов, объединяющих между собой основные устройства ВМ (центральный процессор, память и модули ввода/вывода), образует *структуру взаимосвязей* вычислительной машины. Структура взаимосвязей должна обеспечивать обмен информацией между:

- центральным процессором и памятью;
- центральным процессором и модулями ввода/вывода;
- памятью и модулями ввода/вывода.

Информационные потоки, характерные для основных устройств ВМ, показаны на рис. 4.1.

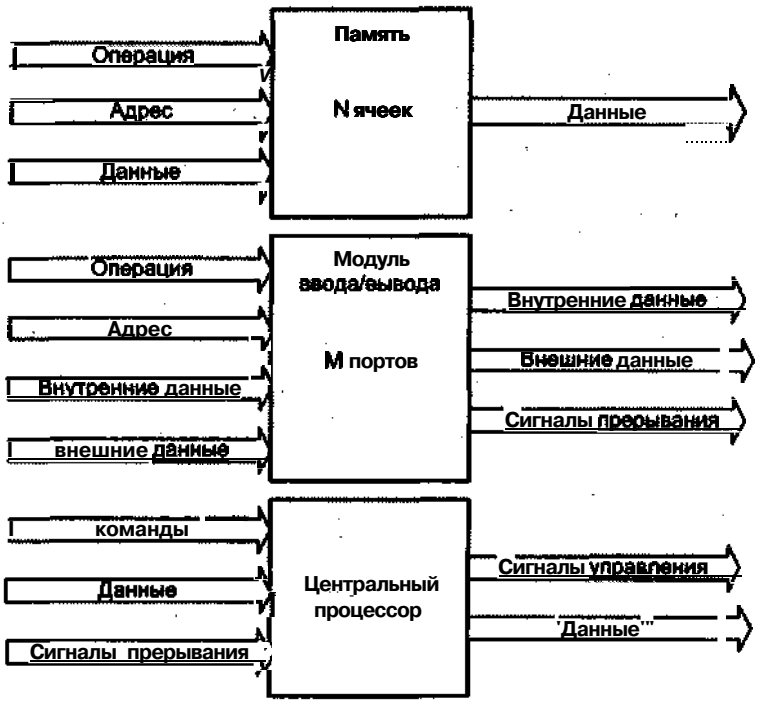


Рис. 4 . 1 . Информационные потоки в вычислительной машине

С развитием вычислительной техники менялась и структура взаимосвязей устройств ВМ (рис. 4.2). На начальной стадии преобладали непосредственные связи между взаимодействующими устройствами ВМ. С появлением мини-ЭВМ, и особенно первых микроЭВМ, все более популярной становится схема с одной общей шиной. Последовавший за этим быстрый рост производительности практически всех устройств ВМ привел к неспособности единственной шины справиться с возросшим трафиком, и ей на смену приходят структуры взаимосвязей на базе нескольких шин. Дальнейшие перспективы повышения производительности вычислений связаны не столько с однопроцессорными машинами, сколько с многопроцессорными вычислительными системами. Способы взаимосвязей в таких системах значительно разнообразнее, и их рассмотрению посвящен один из разделов учебника. Возвращаясь к вычислительным машинам, более внимательно рассмотрим вопросы, связанные с организацией взаимосвязей на базе шин.

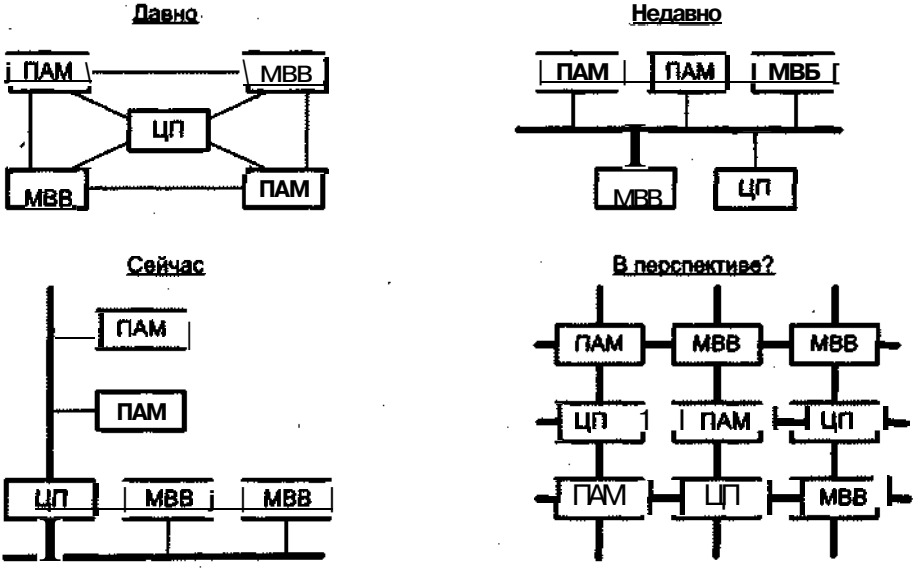


Рис. 4.2. Эволюция структур взаимосвязей (ЦП - центральный процессор, ПАМ •—• модуль основной памяти, МВВ - модуль ввода/вывода)

Взаимосвязь частей ВМ и ее «общение» с внешним миром обеспечиваются системой шин. Большинство машин содержат несколько различных шин, каждая из которых оптимизирована под определенный вид коммуникаций. Часть шин скрыта внутри интегральных микросхем или доступна только в пределах печатной платы. Некоторые шины имеют доступные извне точки, с тем чтобы к ним легко можно было подключить дополнительные устройства, причем большинство таких шин не просто доступны, но и отвечают определенным стандартам, что позволяет подсоединять к шине устройства различных производителей.

Чтобы охарактеризовать конкретную шину, нужно описать (рис. 4.3):

- совокупность сигнальных линий;
- физические, механические и электрические характеристики шины;

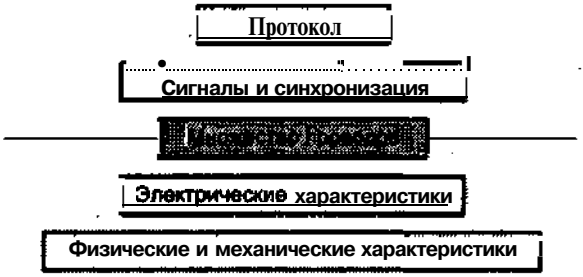


Рис. 4.3. Параметры, характеризующие шину

- используемые сигналы арбитража, состояния, управления и синхронизации;
- правила взаимодействия подключенных к шине устройств (протокол шины).

Шину образует набор коммуникационных линий, каждая из которых способна передавать сигналы, представляющие двоичные цифры 1 и 0. По линии может пересылаться развернутая во времени последовательность таких сигналов. При совместном использовании несколько линий могут обеспечить одновременную (параллельную) передачу двоичных чисел. Физически линии шины реализуются в виде отдельных проводников, как полоски проводящего материала на монтажной плате либо как алюминиевые или медные проводящие дорожки на кристалле микросхемы.

Операции на шине называют *транзакциями.* Основные виды транзакций - *транзакции чтения* и *транзакции записи.* Если в'обмене участвует устройство ввода/вывода, можно говорить о *транзакциях ввода* и *вывода,* по сути эквивалентных транзакциям чтения и записи соответственно. Шинная транзакция включает в себя две части: посылку адреса и прием (или посылку) данных.

Когда два устройства обмениваются информацией по шине, одно из них должно инициировать обмен и управлять им. Такого рода устройства называют *ведущими* (bus master). В компьютерной терминологии «ведущий» - это любое устройство, способное взять на себя владение шиной и управлять пересылкой данных. Ведущий не обязательно использует данные сам. Он, например, может захватить управление шиной в интересах другого устройства. Устройства, не обладающие возможностями инициирования транзакции, носят название *ведомых* (bus slave). В принципе к шине может быть подключено несколько потенциальных ведущих, но в любой момент времени активным может быть только один из них: если несколько устройств передают информацию одновременно, их сигналы перекрываются и искажаются. Для предотвращения одновременной активности нескольких ведущих в любой шине предусматривается процедура допуска к управлению шиной только одного из претендентов (арбитраж). В то же время некоторые шины допускают широковещательный режим записи, когда информация одного ведущего передается сразу нескольким ведомым (здесь арбитраж не требуется). Сигнал, направленный одним устройством, доступен всем остальным устройствам, подключенным к шине.

Английский эквивалент термина «шина» — «bus» — восходит к латинскому слову *omnibus,* означающему «для всего». Этим стремятся подчеркнуть, что шина. ведет себя как магистраль, способная обеспечить всевозможные виды трафика.

В данной главе рассматриваются только общие вопросы, касающиеся организации, функционирования и применения шин, без ориентации на конкретные реализации.

Типы шин

Важным критерием, определяющим характеристики шины, может служить ее целевое назначение. По этому критерию можно выделить:

- шины «процессор-память»;
- шины ввода/вывода;
- системные шины.

Шина «процессор-память»

Шина «процессор-память» обеспечивает непосредственную связь между центральным процессором (ЦП) вычислительной машины и основной памятью (ОП). В современных микропроцессорах такую шину часто называют *шиной переднего плана* и обозначают аббревиатурой FSB (Front-Side Bus). Интенсивный трафик между процессором и памятью требует, чтобы полоса пропускания шины, то есть количество информации, проходящей по шине в единицу времени, была наибольшей. Роль этой шины иногда выполняет системная шина (см. ниже), однако в плане эффективности значительно выгоднее, если обмен между ЦП и ОП ведется по отдельной шине. К рассматриваемому виду можно отнести также шину, связывающую процессор с кэш-памятью второго уровня, известную как *шина заднего плана* - BSB (Back-Side Bus). BSB позволяет вести обмен с большей скоростью, чем FSB, и полностью реализовать возможности более скоростной кэш-памяти.

Поскольку в фон-неймановских машинах именно обмен между процессором и памятью во многом определяет быстродействие ВМ, разработчики уделяют связи ЦП с памятью особое внимание. Для обеспечения максимальной пропускной способности шины «процессор-память» всегда проектируются с учетом особенностей организации системы памяти, а длина шины делается по возможности минимальной.

Шина ввода/вывода

Шина ввода/вывода служит для соединения процессора (памяти) с устройствами ввода/вывода (УВВ). Учитывая разнообразие таких устройств, шины ввода/вывода унифицируются и стандартизируются. Связи с большинством УВВ (но не с видеосистемами) не требуют от шины высокой пропускной способности. При проектировании шин ввода/вывода в учет берутся стоимость конструктива и соединительных разъемов. Такие шины содержат меньше линий по сравнению с вариантом «процессор-Память», но длина линий может быть весьма большой. Типичными примерами подобных шин могут служить шины PCI и SCSI.

Системная шина

С целью снижения стоимости некоторые ВМ имеют общую шину для памяти и устройств ввода/вывода. Такая шина часто называется системной. *Системная шина* служит для физического и логического объединения всех устройств ВМ. Поскольку основные устройства машины, как правило, размещаются на общей монтажной плате, системную шину часто называют объединительной шиной (backplane bus), хотя эти термины нельзя считать строго эквивалентными.

Системная, шина в состоянии содержать несколько сотен линий. Совокупность линий шины можно подразделить на три функциональные группы (рис. 4.4): шину данных, шину адреса и шину управления. К последней обычно относят также линии для подачи питающего напряжения на подключаемые к системной шине модули.

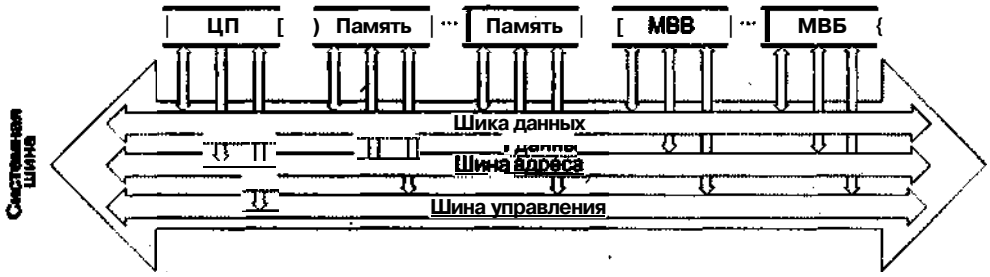


Рис. 4.4. Структура системной шины

Особенности каждой из этих групп и распределение сигнальных линий подробно рассматриваются позже.

Функционирование системной шины можно описать следующим образом. Если один из модулей хочет передать данные в другой, он должен выполнить два действия: получить в свое распоряжение шину и передать по ней данные. Если какой-то модуль хочет получить данные от другого модуля, он должен получить доступ к шине и с помощью соответствующих линий управления и адреса передать в другой модуль запрос. Далее он должен ожидать, пока модуль, получивший запрос, пошлет данные.

Физически системная шина представляет собой совокупность параллельных электрических проводников. Этими проводниками служат металлические полоски на печатной плате. Шина подводится ко всем модулям, и каждый из них подсоединяется ко всем или некоторым ее линиям. Если ВМ конструктивно выполнена на нескольких платах, то все линии шины выводятся на разъемы, которые затем объединяются проводниками на общем шасси.

Среди стандартизированных системных шин универсальных ВМ наиболее известны шины Unibus, Fastbus, Futurebus, VME,NuBus, Multibus-II. Персональные компьютеры, как правило, строятся на основе системной шины в стандартах ISA, EISA или MCA.

Иерархия шин

Если к шине подключено большое число устройств, ее пропускная способность падает, поскольку слишком частая передача прав управления шиной от одного устройства к другому приводит к ощутимым задержкам. По этой причине во многих ВМ предпочтение отдается использованию нескольких шин, образующих определенную иерархию. Сначала рассмотрим ВМ с одной шиной.

Вычислительная машина с одной шиной

В структурах взаимосвязей с одной шиной имеется одна системная шина, обеспечивающая обмен информацией между процессором и памятью, а также между УВВ, с одной стороны, и процессором либо памятью - с другой (рис. 4.5).

Рис. 4.5. Структура взаимосвязей с одной шиной

Для такого подхода характерны простота и низкая стоимость. Однако одношинная организация не в состоянии обеспечить высокие интенсивность и скорость транзакций, причем «узким местом» становится именно шина.

Вычислительная машина с двумя видами шин

Хотя контроллеры устройств ввода/вывода (УВВ) могут быть подсоединены непосредственно к системной шине, больший эффект достигается применением одной или нескольких шин ввода/вывода (рис. 4.6). УВВ подключаются к шинам ввода/вывода, которые берут на себя основной трафик, не связанный с выходом на процессор или память. *Адаптеры шин* обеспечивают буферизацию данных при их пересылке между системной шиной и контроллерами УВВ. Это позволяет ВМ поддерживать работу множества устройств ввода/вывода и одновременно «развязать» обмен информацией по тракту процессор-память и обмен информацией с УВВ.

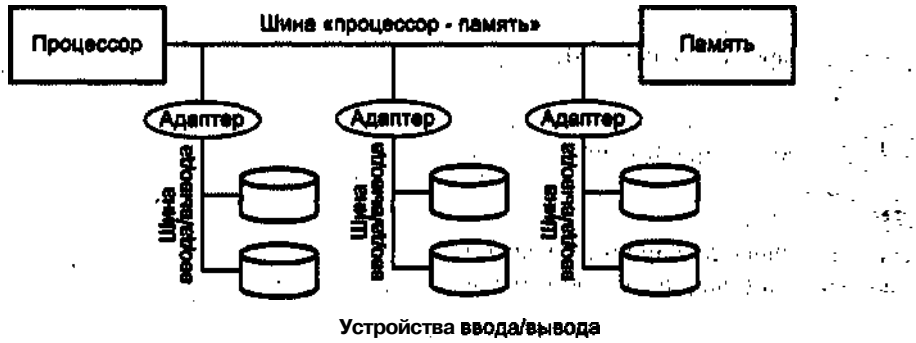


Рис. 4.6. Структура взаимосвязей с двумя видами шин

Подобная схема существенно снижает нагрузку на скоростную шину «процессор-память» и способствует повышению общей производительности ВМ. В качестве примера можно привести вычислительную машину Apple Macintosh II, где роль шины «процессор-память» играет шина NuBus. Кроме процессора и памяти к ней подключаются некоторые УВВ. Прочие устройства ввода/вывода подключаются к шине SCSI Bus.

Вычислительная машина с тремя видами шин

Для подключения быстродействующих периферийных устройств в систему шин может быть добавлена высокоскоростная шина расширения (рис. 4.7).

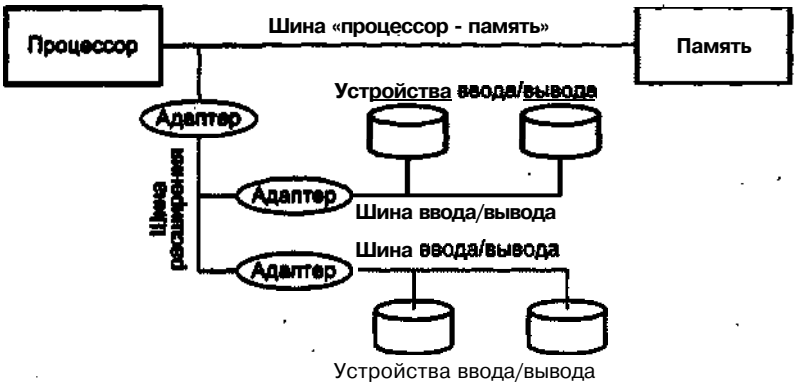


Рис. 4.7. Структура взаимосвязей с тремя видами шин

Шины ввода/вывода подключаются к шине расширения, а уже с нее через адаптер к шине «процессор-память». Схема еще более снижает нагрузку на шину «процессор-память». Такую организацию шин называют *архитектурой с «пристройкой»*(mezzanine architecture).

Физическая реализация шин

Кратко остановимся на различных аспектах физической реализации шин в вычислительных машинах и системах.

Механические аспекты

Основная шина (рис. 4.8), объединяющая устройства вычислительной машины, обычно размещается на так называемой *объединительной* или *материнской плате*. Шину образуют тонкие параллельные медные полоски, поперек которых через небольшие интервалы установлены разъемы для подсоединения устройств ВМ. Подключаемые к шине устройства обычно также выполняются в виде печатных плат, часто называемых *дочерними платами* или *модулями*Дочерние платы вставляются в разъемы на материнской плате. В дополнение к тонким сигнальным линиям на материнской плате имеются также и более широкие проводящие линии, по которым к дочерним платам подводится питающее напряжение. Несколько контактов разъема обычно подключаются к общей точке — «земле». «Земля» на мате-

ринской плате реализуется либо в виде медного слоя (одного из внутренних слоев многослойной печатной платы), либо как широкая медная дорожка на обратной стороне материнской платы.

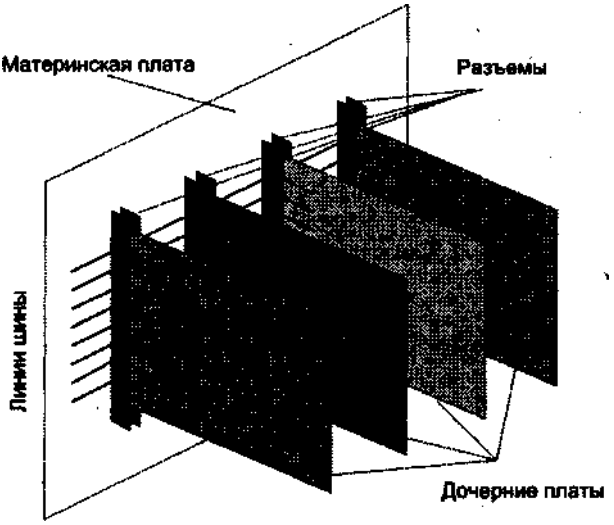


Рис. 4.8. Организация объединительной шины

Контактные пружины в разъемах обеспечивают независимое подключение сигнальных линий, расположенных по обеим сторонам вставляемой в разъем дочерней платы. При создании соединительных разъемов прилагаются значительные усилия с тем, чтобы гарантировать надежный контакт после многократного извлечения платы из разъема, а также при длительной (многолетней) эксплуатации разъема в загрязненной или коррозийной среде.

«Механические» спецификации шины обычно включают такие детали, как размеры плат, размеры и размещение направляющих для установки платы, разрешенное место для установки кабельного разъема, максимальная высота элементов на плате и т. д.

Электрические аспекты

Все устройства, использующие шину, электрически подсоединены к ее сигнальным линиям, представляющим собой электрические проводники. Меняя уровни напряжения на сигнальных линиях, ведущее устройство формирует **на** них информационные или управляющие сигналы. Когда ведущее устройство выставляет на сигнальной шине какой-то уровень напряжения, этот уровень может быть воспринят приемниками в любой точке линии. Такое описание дает лишь идеализированную картину происходящих на шине процессов — реальные процессы значительно, сложнее.

Схему, меняющую напряжение на сигнальной шине, обычно называют *драйвером* или *возбудителем щины.* В принципе драйвером может быть любая цифровая схема, поскольку на ее цифровом выходе всегда присутствует один из двух возможных уровней напряжения.

При реализации шины необходимо предусмотреть возможность отключения драйвера от сигнальной линии на период, когда он не использует шину. Один из возможных способов обеспечения подобного режима - применение драйвера, выход которого может находиться в одном из трех состояний: «высокий уровень напряжения» (high), «низкий уровень напряжения» (low) и «отключен» (off). Для перевода в состояние «off», эквивалентное отключению выхода драйвера от сигнальной линии, используется специальный вход драйвера. Режим «off» необходим для исключения возможности одновременного управления шиной двумя или более устройствами, в противном случае на линиях могут возникать пиковые выбросы напряжения или искаженные сигналы, которые кроме некорректной передачи информации могут привести к преждевременному отказу электронных компонентов.

Совместное использование линии шины несколькими устройствами возможно также за счет подключения этой линии к выходу драйвера через резистор, соединенный с источником питания. В зависимости от полупроводниковой технологии, примененной в выходных каскадах драйвера, подобную возможность обеспечивают схемы с открытым коллектором (ТТЛ), открытым стоком (МОП) или открытым эмиттером (ЭСЛ). Данный способ не только исключает электрические конфликты на шине, но и позволяет реализовать очень полезный вид логической операции, известный как «монтажное ИЛИ» или «монтажное И» (трактовка зависит от соответствия между уровнями напряжения и логическими значениями 1 и 0). Если к линии одновременно подключается несколько драйверов, то сигнал на линии представляет собой результат логического сложения (операция ИЛИ) всех поступивших на линию сигналов. Это оказывается весьма полезным при решении задачи арбитража, которая рассматривается позже. В некоторых шинах «монтажное ИЛИ» используется лишь в отдельных сигнальных линиях, но иногда эту операцию допускают по отношению ко всем линиям шины.

Приемниками в операциях на шинах называют схемы, сравнивающие уровень сигнала на входе со стандартными значениями, формируемыми внутренними цепями приемников. По итогам сравнения приемник генерирует выходной сигнал, уровень которого соответствует одному из двух возможных логических значений — 1 или 0. *Трансивер* (приемопередатчик) содержит приемник и драйвер, причем выход драйвера и вход приемника сводятся в общую точку.

Рассматривая процесс распространения сигнала по сигнальной линии, необходимо учитывать четыре основных фактора:

- скорость распространения;
- отражение;
- перекос;
- эффекты перекрестного влияния.

Теоретическая граница скорости распространения сигнала — скорость света в свободном пространстве, то есть около 300 мм/нс. Реальная скорость, определяемая физическими характеристиками сигнальных линий и нагрузкой, реально не может превысить 70% от скорости света.

Процессы в линии рассмотрим на примере сигнальной линии, которая через резистор, соединенный с источником питания, удерживается на уровне напряже-

ния, соответствующем логической единице. Сигнал драйвера «подтягивает» линию к своему уровню напряжения. Изменение напряжения распространяется от точки подключения драйвера в обоих направлениях, пока на всей линии не установится уровень сигнала драйвера. Характер распространения сигнала определяют емкость, индуктивность и характеристическое сопротивление линии, локальные значения которых по длине линии зависят от локальных свойств проводника и его окружения.

По мере распространения по реальной линии сигнал преодолевает области с различным сопротивлением. Там, где оно меняется, сигнал не может оставаться постоянным, поскольку меняется соотношение между током и напряжением. Часть сигнала продолжает продвижение, а часть — отражается в противоположную сторону. Прямой и отраженный сигналы могут повторно отражаться, в результате чего на линии формируется сложный результирующий сигнал. В конце линии сигнал отражается назад, если только он не поглощен правильно подобранным согласующим резистором. Если на конце линии имеется согласующий резистор, с сопротивлением, идентичным импедансу линии, сигнал будет поглощен без отражения. Такие резисторы должны размещаться по обоим концам сигнальной линии. К сожалению, точное значение импеданса реальной линии никогда не известно, из-за чего номиналы резисторов невозможно точно согласовать с линией, и отражение всегда имеет место.

При параллельной передаче по линиям шины битов адреса или данных сигналы на разных линиях достигают соответствующих приемников совсем не одновременно. Это явление известно как *перекос сигналов*. Причины возникновения и способы компенсации перекоса будут рассмотрены позже.

Распространяясь по линии, сигнал создает вокруг нее электростатическое и магнитное поля. Сигнальные линии в шине располагаются параллельно и в непосредственной близости одна от другой. Поля от близко расположенных линий перекрываются, приводя к тому, что сигнал на одной линии влияет на сигнал в другой. Этот эффект называют *перекрестной* или *переходной помехой*.

Наиболее очевидный способ уменьшения перекрестной помехи эффекта - пространственно разнести линии шины так, чтобы их поля не влияли на «соседей» — для печатной платы ограниченного размера не подходит. К снижению эффектов перекрестного влияния ведет уменьшение взаимных емкости и индуктивности линий, чего можно добиться, разместив вблизи сигнальных линий «земляные» линии или включив в многослойную печатную плату «земляные» слои. Это, однако, приводит к нежелательному эффекту увеличения собственной емкости линий. Наиболее распространенный подход к снижению перекрестной помехи состоит в разделении линий изолятором с малой диэлектрической постоянной. В целом, при проектировании шин обычно используется комбинация перечисленных методов борьбы с перекрестной помехой.

Из-за несовершенства физической реализации сигнальных линий фронты импульсов по мере распространения сигналов меняются, соответственно, меняется и форма сигнала. Для каждой шины существует некое минимальное значение ширины импульса, при которой он еще способен дойти от одного конца к другому так, что его еще можно распознать. Эта ширина выступает в качестве основного

ограничения на полосу пропускания данной шины, то есть на число импульсов, которые могут быть переданы по шине в единицу времени.

Поскольку драйвер одновременно «видит» две линии, передающие информацию в противоположных направлениях, он должен поддерживать двойную по сравнению с одной линией величину тока. Для типичных линий импеданс не превышает 20 Ом, а сигналы имеют уровень порядка 3 В, что выражается в величине тока около 150 мА. Приведенные цифры для современных драйверов не составляют проблемы, поскольку применяемые в настоящее время схемы способны приспособиться к гораздо худшим параметрам сигналов.

Порождаемый сигналом ток замыкается через «земляной» контакт драйвера. Когда одновременно активны все сигнальные линии, ток возврата через «землю» может быть весьма большим. Положение осложняет то, что ток этот не является постоянным и в моменты подключения и отключения драйвера содержит высокочастотные составляющие. Кроме того, из-за сопротивления и индуктивности «земляного» слоя печатной платы потенциалы на «земляных» выводах дочерних плат могут различаться. Это может приводить к неверной оценке сигналов приемниками, следствием чего становится некорректное срабатывание логических схем. С «земляным» шумом легче бороться на стадии проектирования шины. Прежде всего необходимо улучшать характеристики «земляных» слоев на материнской и дочерних платах. Между системами заземления материнской и дочерних плат должно быть много хорошо распределенных надежных контактов. Для высокоскоростных шин на каждые четыре сигнальных шины следует иметь отдельный «земляной» контакт. Кроме того, дочерняя плата должна быть спроектирована так, чтобы «земляной» ток от данного драйвера протекал к тому «земляному» контакту, который расположен как можно ближе к сигнальным выводам. «Земля» материнской платы обычно реализуется в виде внутреннего медного слоя в многослойной печатной плате; отверстия с зазором вокруг сигнальных выводов предотвращают короткое замыкание сигнального вывода с этим слоем. Разъем должен быть достаточно широким, чтобы на дочерней плате трансиверы можно было разместить по возможности ближе к нему, что позволяет сократить длину тех участков шины, где нарушается ее неразрывность.

В целом ряде известных шин многие из рассмотренных положений игнорируются. По практическим соображениям используются линии с высоким импедансом. Надежность работы с такими «плохими» шинами достигается за счет их замедления: затягивание перехода сигналов от одного уровня напряжения к другому приводит к уменьшению отражений. Снижается также влияние перекрестных помех.

Высокое быстродействие драйверов шины имеет и отрицательную сторону: они оказываются слишком быстрыми для управляемых ими шин, при этом сигналы на линиях сильно искажаются. Эта проблема обычно преодолевается за счет введения задержки, часто называемой *временем установления сигнала* (временем успокоения). Задержка выбирается так, что сигналы стабилизируются до момента их использования. Зачастую достаточно задержки, принципиально присущей используемым схемам, но иногда приходится вводить и явную задержку.

В синхронных шинах, где для синхронизации транзакций используется единая система тактовых импульсов (ТИ), такая задержка может быть добавлена весьма

просто путем замедления тактирования. Так, можно разрешить всем сигналам изменяться только по одному из фронтов ТИ, что создает достаточную заминку для распространения сигналов и их стабилизации.

В асинхронных шинах проблема должна быть решена либо в самом драйвере, либо за счет введения искусственной приостановки, компенсирующей излишнее быстродействие драйвера. Еще одна возможность - замедление цепей приемника.

Чтобы сделать приемники нечувствительными к отражениям и высокочастотному шуму, в них встраивают фильтры нижних частот. В шине NITS Altair, например, используются драйверы большой мощности и маломощные приемники. По причине быстрых драйверов и неудачного дизайна монтажной шины сигналы в этой шине сильно искажаются, но маломощные приемники достаточно медлительны и позволяют нивелировать большинство из дефектов сигнала.

Применяющиеся в настоящее время драйверы и приемники на базе транзисторно-транзисторной логики (ТТЛ) уже не в полной мере отвечают растущим требованиям. В новых шинах наметилась тенденция перехода к трансиверам на основе эмиттерно-связанной логики (ЭСЛ), как, например, в шине Fastbus. Замечательно, что одновременно с уменьшением емкости линий, уровней и крутизны фронтов сигналов, подавлением шумов в приемнике, в подобных трансиверах сохраняется преемственность со старыми устройствами: они допускают использование со стороны дочерних плат источников питания и сигналов, характерных для ТТЛ-технологии.

Обычно перед установкой или извлечением дочерней платы требуется отключение источника питания машины. В мультипроцессорных системах это крайне нежелательно, поскольку временное отключение питания приводит к необходимости перезагрузки и перезапуска каждого процессора. Некоторые системы проектируются так, что допускают извлечение и установку платы в присутствии питающего напряжения. В них обеспечивается сохранение состояния остальных плат, но работа шины временно приостанавливается. Естественно, что плата, которая была удалена и заменена на другую, уже не находится в исходном состоянии и должна быть инициализирована. Чаще всего реализация подобного режима оказывается чересчур дорогостоящей.

Распределение линий шины

Любая транзакция на шине начинается с выставления ведущим устройством адресной информации. Адрес позволяет выбрать ведомое устройство и установить соединение между ним и ведущим. Для передачи адреса используется часть сигнальных линий шины, совокупность которых часто называют *шиной адреса (ША)*.

На ША могут выдаваться адреса ячеек памяти, номера регистров ЦП, адреса портов ввода/вывода и т. п. Многообразие видов адресов предполагает наличие дополнительной информации, уточняющей вид, используемый в данной транзакции. Такая информация может косвенно содержаться в самом адресе, но чаще передается по специальным управляющим линиям шины.

Разнообразной может быть и структура адреса. Так, в адресе может конкретизироваться лишь определенная часть ведомого, например, старшие биты адреса

могут указывать на один из модулей основной памяти, в то время как младшие биты определяют ячейку внутри этого модуля.

В некоторых шинах предусмотрены адреса специального вида, обеспечивающие одновременный выбор определенной группы ведомых либо всех ведомых сразу (broadcast). Такая возможность обычно практикуется в транзакциях записи (от ведущего к ведомым), однако существует также специальный вид транзакции чтения (одновременно от нескольких ведомых общему ведущему). Английское название такой транзакции чтения *broadcall* можно перевести как *«широковещательный опрос»*. Информация, возвращаемая ведущему, представляет собой результат побитового логического сложения данных, поступивших от всех адресуемых ведомых.

Число сигнальных линий, выделенных для передачи адреса *{ширина шины адреса),* определяет максимально возможный размер адресного пространства. Это одна из базовых характеристик шины, поскольку от нее зависит потенциальная емкость адресуемой памяти и число обслуживаемых портов ввода/вывода.

Совокупность линий, служащих для пересылки данных между модулями системы, называют *шиной данных* (ШД). Важнейшие характеристики шины данных - ширина и пропускная способность.

Ширина шины данных определяется количеством битов информации, которое может быть передано по шине за одну транзакцию *{цикл шины).* Цикл шины следует отличать от периода тактовых импульсов — одна транзакция на шине может занимать несколько тактовых периодов. В середине 1970-х годов типовая ширина шины данных составляла 8 бит. В наше время это обычно 32,64 или 128 бит. В любом случае ширину шины данных выбирают кратной целому числу байтов, причем это число, как правило, представляет собой целую степень числа 2.

Элемент данных, задействующий всю ширину ШД, принято называть *словом,* хотя в архитектуре некоторых ВМ понятие «слово» трактуется по-другому, то есть слово может иметь разрядность, не совпадающую с шириной ШД.

В большинстве шин используются адреса, позволяющие указать отдельный байт слова. Это свойство оказывается полезным, когда желательно изменить в памяти лишь часть полного слова.

При передаче по ШД части слова пересылка обычно производится по тем же сигнальным линиям, что и в случае пересылки полного слова, однако в ряде шин «урезанное» слово передается по младшим линиям ШД. Последний вариант может оказаться более удобным при последующем расширении шины данных, поскольку в этом случае сохраняется преемственность со «старой» шиной.

Ширина шины данных существенно влияет на производительность ВМ. Так, если шина данных имеет ширину вдвое меньшую чем длина команды, ЦП в течение каждого цикла команды вынужден осуществлять доступ к памяти дважды.

Пропускная способность шины характеризуется количеством единиц информации (байтов), которые допускается передать по шине за единицу времени (секунду), а. определяется физическим построением шины и природой подключаемых к ней устройств. Очевидно, что чем шире шина, тем выше ее пропускная способность.

Последовательность событий, происходящих на шине данных в процессе одной транзакции, иллюстрирует рис. 4.9. Пусть устройство А на одном конце шины передает данные устройству В на другом ее конце.

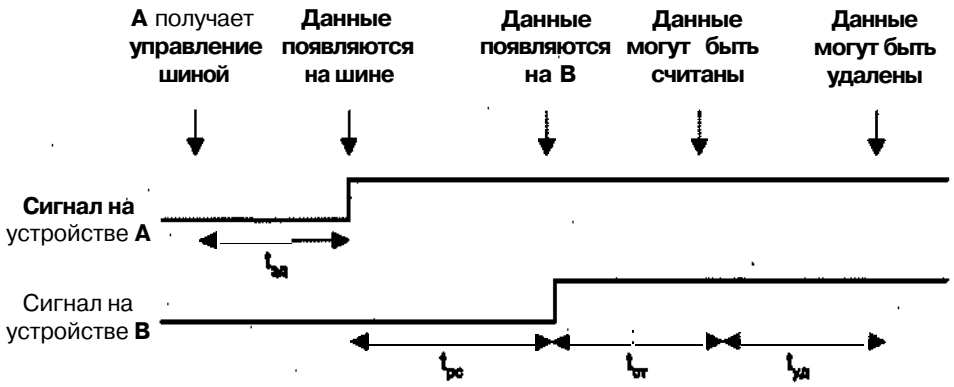


Рис. 4.9. Временная диаграмма пересылки данных

Сначала устройство А выставляет данные на шину. Здесь $t_{зд}$ — это задержка между моментом выставления данных устройством А и моментом их появления на шине. Этот интервал времени может составлять от 1 до 4 нс. Как уже отмечалось, скорость распространения данных по шине реально не в состоянии превысить 70% от скорости света. Единственный способ уменьшения задержки распространения $t_{рс}$ — сокращение длины шины. Когда сигнал достигает устройства, он должен быть «захвачен». Захват данных устройством В может быть произведен только по прошествии некоторого времени стабилизации. Время стабилизации $t_{ст}$ — это время, в течение которого данные на входе устройства В должны стабилизироваться с тем, чтобы их можно было однозначно распознать. Необходимо также упомянуть и время удержания $t_{уд}$ — интервал, в течение которого информация должна оставаться на шине данных после того, как они были зафиксированы устройством В.

Общее время передачи данных по шине $t_{п}$ определяется выражением $t_{п} = t_{зд} + t_{рс} + t_{ст} + t_{уд}$. Если подставить типовые значения этих параметров, получим 4 + 1,5 + + 2 + 0 = 7,5 нс, что соответствует частоте шины 109/7,5 = 133,3 МГц.

На практике передача данных осуществляется с задержкой на инициализацию транзакции ($t_{и}$). Учитывая эту задержку, максимальную скорость передачи можно определить как $1/(t_{П} + t_{И})$

Некоторые шины содержат дополнительные линии, используемые для обнаружения ошибок, возникших в процессе передачи. Выделение по одной дополнительной линии на каждый отдельный байт данных позволяет контролировать любой байт по паритету, причем и в случае пересылки по ШД лишь части слова. Возможен и иной вариант контроля ошибок. В этом случае упомянутые дополнительные линии используются совместно. По ним передается корректирующий код, благодаря которому ошибка может быть не только обнаружена, но и откорректирована. Такой метод удобен лишь при пересылке по шине полных слов.

Если адрес и данные в шине передаются по независимым (выделенным) сигнальным линиям, то ширина ШАи ШД обычно выбирается независимо. Наиболее частые комбинации: 16-8,16-16, 20-8, 20-16, 24-32 и 32-32. Во многих шинах адрес и данные пересылаются по одним и тем же линиям, но в разных тактах цикла шины. Этот прием называется *временным мультиплексированием и* будет

рассмотрен позже. Здесь же отметим, что в случае мультиплексирования ширина ША и ширина ШД должны быть взаимоувязаны.

Применение раздельных шин адреса и данных позволяет повысить эффективность использования шины, особенно в транзакциях записи, поскольку адрес ячейки памяти и записываемые данные *могут* передаваться одновременно.

Помимо трактов пересылки адреса и данных, неотъемлемым атрибутом любой шины являются линии, по которым передается управляющая информации и информация о состоянии участвующих в транзакции устройств. Совокупность таких линий принято называть *шиной управления* (ШУ), хотя такое название представляется не совсем точным. Сигнальные линии, входящие в ШУ, можно условно разделить на несколько групп.

Первую группу образуют линии, по которым пересылаются *сигналы управления транзакциями,* то есть сигналы, определяющие:

- тип выполняемой транзакции (чтение или запись);
- количество байтов, передаваемых по шине данных, и, если пересылается часть слова, то какие байты;
- какой тип адреса выдан на шину адреса;
- какой протокол передачи должен быть применен.

На перечисленные цели обычно отводится от двух до восьми сигнальных линий.

Ко второй группе отнесем линии передачи *информации состояния (статуса).* В эту группу входят от одной до четырех линий, по которым ведомое устройство может информировать ведущего о своем состоянии или передать код возникшей ошибки.

Третья группа - *линии арбитража.* Вопросы арбитража рассматриваются несколько позже. Пока отметим лишь, что арбитраж необходим для выбора одного из нескольких ведущих, одновременно претендующих на доступ к шине. Число линий арбитража в разных шинах варьируется от 3 до 11.

Четвертую группу образуют *линии прерывания.По* этим линиям передаются запросы на обслуживание, посылаемые от ведомых устройств к ведущему. Под собственно запросы обычно отводятся одна или две линии, однако при одновременном возникновении запросов от нескольких ведомых возникает проблема арбитража, для чего могут понадобиться дополнительные линии, если только с этой целью не используются линии третьей группы.

Пятая группа - линии для организации *последовательных локальных сетей.* Наличие от 1 до 4 таких линий стало общепринятой практикой в современных шинах. Обусловлено это тем, что последовательная передача данных протекает значительно медленнее, чем параллельная, и сети значительно выгоднее строить, не загружая быстрые линии основных шин адреса и данных. Кроме того, шины этой группы могут быть использованы как полноценный, хотя и медленный, избыточный тракт для замены ША и ШД в случае их отказа. Иногда шины пятой группы назначаются для реализации специальных функций, таких, например, как обработка прерываний или сортировка приоритетов задач.

В некоторых ШУ имеется шестая группа сигнальных линий — от 4 до 5 *линий позиционного кода,* подсоединяемых к специальным выводам разъема С помощью

перемычек на этих выводах можно задать уникальный позиционный код разъема на материнской плате или вставленной в этот разъем дочерней платы. Такой код может быть использован для индивидуальной инициализации каждой отдельной платы при включении или перезапуске системы.

Наконец, в каждой шине обязательно присутствуют линии, которые в нашей классификации входят в **седьмую группу,** которая по сути является одной из важнейших. Это группа линий *тактирования и синхронизации.* При проектировании шины таким линиям уделяется особое внимание. В состав группы, в зависимости от протокола шины (синхронный или асинхронный), входят от двух до шести линий.

В довершение необходимо упомянуть линии для подвода питающего напряжения и линии заземления.

Большое количество линий в шине предполагает использование разъемов со значительным числом контактов. В некоторых шинах разъемы имеют сотни контактов, где предусмотрены подключение вспомогательных шин специального назначения, свободные линии для локального обмена между дочерними платами, множественные параллельно расположенные контакты для «размножения» питания и «земли». Значительно чаще число контактов разъема ограничивают. В табл. 4.1 показано возможное распределение линий 32-разрядной шины в 64-контактном разъеме.

Таблица 4 . 1 . Распределение линий 32-разрядной шины в 64-контактном разъеме

Линии	Типовое число выводов	Комментарии
Адрес	16-32	Могут быть объединены в мультиплексируемую шину
Данные	8-32	
Арбитраж	3-11	
Управление	2-8	
Состояние	1-4	
Тактирование и синхронизация	2-6	
Локальная сеть	1-4	
Позиционный код	4-5	
Питание	2-20	
«Земля»	2-20	

Выделенные и мультиплексируемые линии

В некоторых ВМ линии адреса и данных объединены в единую *мультиплексируемую шину адреса/данных.* Такая шина функционирует в режиме разделения времени, поскольку цикл шины разбит на временной интервал для передачи адреса и временной интервал для передачи данных. Структура такой шины показана на рис. 4.10.

Мультиплексирование адресов и данных предполагает наличие мультиплексора на одном конце тракта пересылки информации и демультиплексора на его дру-

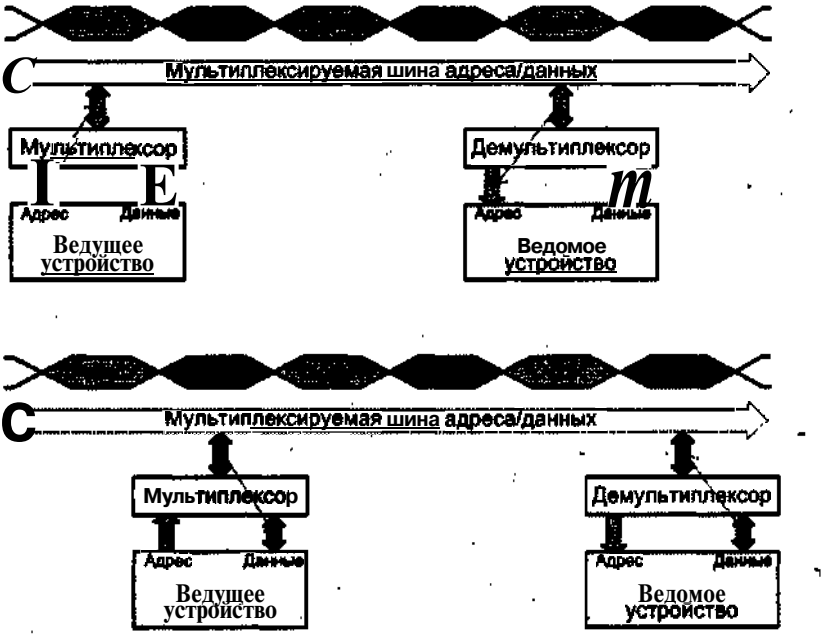


Рис. 4.10. Мультиплексирование адреса и данных

гом конце. Мультиплексоры и демультиплексоры играют роль коммутирующих устройств.

Мультиплексирование позволяет сократить общее число линий, но требует усложнения логики связи с шиной. Кроме того, оно ведет к потенциальному снижению производительности, поскольку исключает возможность параллельной передачи адресов и данных, что можно было бы использовать в транзакциях записи, одновременно выставляя на ША адрес, а на ШД — записываемое слово.

Примером применения мультиплексируемой шины адреса/данных может служить шина Futurebus+.

Арбитраж шин

В реальных системах на роль ведущего вправе одновременно претендовать сразу несколько из подключенных к шине устройств, однако управлять шиной в каждый момент времени может только одно из них. Чтобы исключить конфликты, шина должна предусматривать определенные механизмы арбитража запросов и правила предоставления шины одному из запросивших устройств. Решение обычно принимается на основе приоритетов претендентов.

Схемы приоритетов

Каждому потенциальному ведущему присваивается определенный уровень приоритета, который может оставаться неизменным *(статический* или *фиксированный приоритет)* либо изменяться по какому-либо алгоритму *(динамический приоритет)*.

Основной недостаток статических приоритетов в том, что устройства, имеющие высокий приоритет, в состоянии полностью блокировать доступ к шине устройств с низким уровнем приоритета. Системы с динамическими приоритетами дают шанс каждому из запросивших устройств рано или поздно получить право на управление шиной, то есть в таких системах реализуется принцип равнодоступности.

Наибольшее распространение получили следующие алгоритмы динамического изменения приоритетов:

- простая циклическая смена приоритетов;
- циклическая смена приоритетов с учетом последнего запроса;
- смена приоритетов по случайному закону;
- схема равных приоритетов;
- алгоритм наиболее давнего использования.

В алгоритме *простой циклической смены приоритетов* после каждого цикла арбитража все приоритеты понижаются на один уровень, при этом устройство, имевшее ранее низший уровень приоритета, получает наивысший приоритет.

В схеме *циклической смены приоритетов с учетом последнего запроса* все возможные запросы упорядочиваются в виде циклического списка. После обработки очередного запроса обслуженному ведущему назначается низший уровень приоритета. Следующее в списке устройство получает наивысший приоритет, а остальным устройствам приоритеты назначаются в убывающем порядке, согласно их следованию в циклическом списке.

В обеих схемах циклической смены приоритетов каждому ведущему обеспечивается шанс получить шину в свое распоряжение, однако большее распространение получил второй алгоритм.

При *смене приоритетов по случайному закону* после очередного цикла арбитража с помощью генератора псевдослучайных чисел каждому ведущему присваивается случайное значение уровня приоритета.

В *схеме равных приоритетов* при поступлении к арбитру нескольких запросов каждый из них имеет равные шансы на обслуживание. Возможный конфликт разрешается арбитром. Такая схема принята в асинхронных системах.

В *алгоритме наиболее давнего использования* (LRU, Least Recently Used) после каждого цикла арбитража наивысший приоритет присваивается ведущему, который дольше чем другие не использовал шину.

Помимо рассмотренных существует несколько алгоритмов смены приоритетов, которые не являются чисто динамическими, поскольку смена приоритетов происходит не после каждого цикла арбитража. К таким алгоритмам относятся:

- алгоритм очереди (первым пришел — первым обслужен);
- алгоритм фиксированного кванта времени.

В *алгоритме очереди* запросы обслуживаются в порядке очереди, образовавшейся к моменту начала цикла арбитража. Сначала обслуживается первый запрос в очереди, то есть запрос, поступивший раньше остальных. Аппаратурная реализация алгоритма связана с определенными сложностями, поэтому используется он редко.

В *алгоритме фиксированного кванта времени* каждому ведущему для захвата шины в течение цикла арбитража выделяется определенный квант времени. Если ведущий в этот момент не нуждается в шине, выделенный ему квант остается не использованным. Такой метод наиболее подходит для шин с синхронным протоколом.

Схемы арбитража

Арбитраж запросов на управление шиной может быть организован по централизованной или децентрализованной схеме. Выбор конкретной схемы зависит от требований к производительности и стоимостных ограничений.

Централизованный арбитраж

При *централизованном арбитраже* в системе имеется специальное устройство — *центральный арбитр,* - ответственное за предоставление доступа к шине только одному из запросивших ведущих. Это устройство, называемое иногда *центральным контроллером шины,* может быть самостоятельным модулем или частью ЦП. Наличие на шине только одного арбитра означает, что в централизованной схеме имеется единственная точка отказа. В зависимости от того, каким образом ведущие устройства подключены к центральному арбитрут возможные схемы централизованного арбитража можно подразделить на параллельные и последовательные.

В параллельном варианте центральный арбитр связан с каждым потенциальным ведущим индивидуальными двухпроводными трактами. Поскольку запросы к центральному арбитру могут поступать независимо и параллельно, данный вид арбитража называют *централизованным параллельным арбитражем* или *централизованным арбитражем независимых запросов.*

Идею централизованного параллельного арбитража на примере восьми ведущих устройств иллюстрирует рис. 4.11, *а.*

Здесь и далее под «текущим ведущим» будем понимать ведущее устройство, управляющее шиной в момент поступления нового запроса. Устройство, выставившее запрос на управление шиной, будем называть «запросившим ведущим». Сигналы запроса шины (ЗШ) поступают на вход центрального арбитра по индивидуальным линиям. Ведущему с номером *i,* который был выбран арбитром, также по индивидуальной линии возвращается сигнал предоставления шины (ПШ,). Реально же занять шину новый ведущий сможет лишь после того, как текущий ведущий (пусть он имеет номер j) снимет сигнал занятия шины (ШЗ). Текущий ведущий должен сохранять сигналы ШЗ и ЗШj активными в течение всего времени, пока он использует шину. Получив запрос от ведущего, приоритет которого выше, чем у текущего ведущего, арбитр снимает сигнал ПШj на входе текущего ведущего и выдает сигнал предоставления шины ПШ, запросившему ведущему. В свою очередь, текущий ведущий, обнаружив, что центральный арбитр убрал с его входа сигнал ПШj , снимает свои сигналы ШЗ и ЗШj после чего запросивший ведущий может перенять управление шиной. Если в момент пропадания сигнала ПШ на шине происходит передача информации, текущий ведущий сначала завершает передачу и лишь после этого снимает свои сигналы.

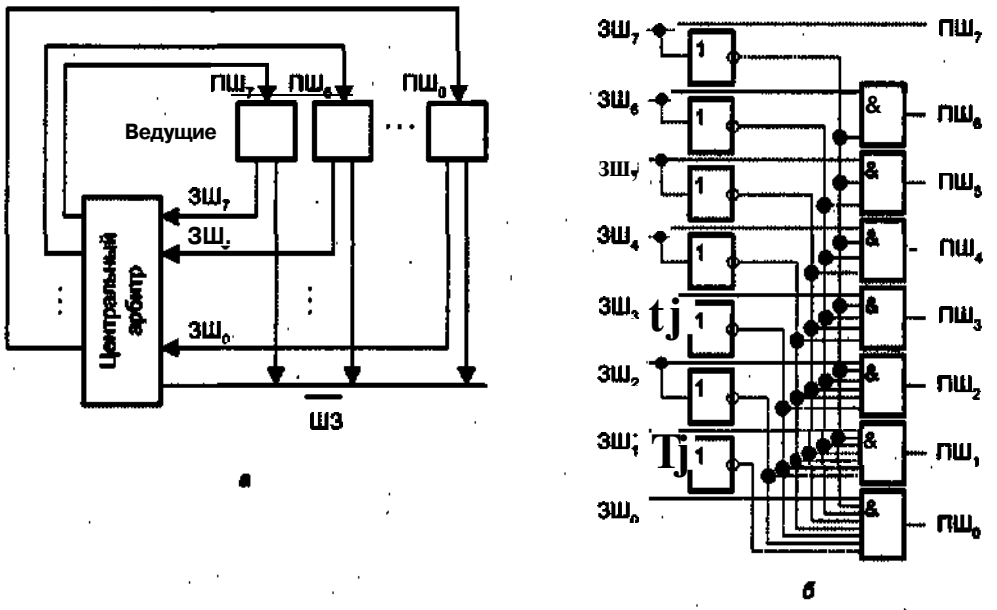


Рис. 4.11. Централизованный параллельный арбитраж: а - общая схема; *б* — возможная реализация

Логика выбора одного из запрашивающих ведущих обычно реализуется аппаратными средствами. В качестве примера рассмотрим реализацию системы централизованного параллельного арбитража для статических приоритетов (рис. 4.11, б). Пусть имеется восемь потенциальных ведущих 7-0, восемь сигналов запроса шины $ЗШ_7$-$ЗШ_0$ и восемь соответствующих им сигналов предоставления шины $ПШ_7$-$ПШ_0$. Положим, что приоритеты ведущих последовательно убывают с уменьшением их номера. Если текущим является ведущий 3, то шину у него могут перехватить ведущие с номерами от 4 до 7, а ведущие 0-2 этого сделать не могут. Ведущий 0 вправе использовать шину лишь тогда, когда она свободна, и должен освободить ее по запросу любого другого ведущего. Схема статических приоритетов может быть относительно просто реализована на основе логических выражений, которые применительно к рассматриваемому примеру имеют вид:

$$ПШ_7 = ЗШ_7;$$
$$ПШ_6 = \overline{ЗШ_7} \cdot ЗШ_6;$$
$$ПШ_5 = \overline{ЗШ_7} \cdot \overline{ЗШ_6} \cdot ЗШ_5;$$
$$ПШ_4 = \overline{ЗШ_7} \cdot \overline{ЗШ_6} \cdot \overline{ЗШ_5} \cdot ЗШ_4;$$
$$ПШ_3 = \overline{ЗШ_7} \cdot \overline{ЗШ_6} \cdot \overline{ЗШ_5} \cdot \overline{ЗШ_4} \cdot ЗШ_3;$$
$$ПШ_2 = \overline{ЗШ_7} \cdot \overline{ЗШ_6} \cdot \overline{ЗШ_5} \cdot \overline{ЗШ_4} \cdot \overline{ЗШ_3} \cdot ЗШ_2;$$
$$ПШ_1 = \overline{ЗШ_7} \cdot \overline{ЗШ_6} \cdot \overline{ЗШ_5} \cdot \overline{ЗШ_4} \cdot \overline{ЗШ_3} \cdot \overline{ЗШ_2} \cdot ЗШ_1;$$
$$ПШ_0 = \overline{ЗШ_7} \cdot \overline{ЗШ_6} \cdot \overline{ЗШ_5} \cdot \overline{ЗШ_4} \cdot \overline{ЗШ_3} \cdot \overline{ЗШ_2} \cdot \overline{ЗШ_1} \cdot ЗШ_0.$$

Устройства арбитража, реализующие систему статических приоритетов, обычно выполняются в виде отдельных микросхем (например, SN74278 фирмы Texas

Instruments), которые, с целью увеличения числа входов и выходов, могут объединяться по каскадной схеме, что, однако, ведет к увеличению времени арбитража.

При наличии большого числа источников запроса центральный арбитр может строиться по схеме двухуровневого параллельного арбитража. Все возможные запросы разбиваются на группы, и каждая группа анализируется своим арбитром первого уровня. Каждый арбитр первого уровня выбирает запрос, имеющий в данной группе наивысший приоритет. Арбитр второго уровня отдает предпочтение среди арбитров первого уровня, обнаруживших запросы на шину, тому, который имеет более высокий приоритет. Если количество возможных запросов очень велико, могут вводиться дополнительные уровни арбитража.

Схема централизованного параллельного арбитража очень гибка — вместо статических приоритетов допускается использовать любые варианты динамической смены приоритетов. Благодаря наличию прямых связей между центральным арбитром и ведущими схема обладает высоким быстродействием, однако именно непосредственные связи становятся причиной повышенной стоимости реализации. В параллельных схемах затруднено подключение дополнительных устройств. Обычно максимальное число ведущих при параллельном арбитраже не превышает восьми. У схемы есть еще один существенный недостаток — сигналы запроса и подтверждения присутствуют только на индивидуальных линиях и не появляются на общих линиях шины, что затрудняет диагностику.

Второй вид централизованного арбитража известен как *централизованный последовательный арбитраж*. В последовательных схемах для выделения запроса с наивысшим приоритетом используется один из сигналов, поочередно проходящий через цепочку ведущих, чем и объясняется другое название - *цепочечный* или *гирляндный арбитраж*. В дальнейшем будем полагать, что уровни приоритета ведущих устройств в цепочке понижаются слева направо.

В зависимости от того, какой из сигналов используется для целей арбитража, различают три основных типа схем цепочечного арбитража: с цепочкой для сигнала предоставления шины, с цепочкой для сигнала запроса шины (ЗШ) и с цепочкой для дополнительного сигнала разрешения (РШ). Наиболее распространена схема *цепочки для сигнала ПШ* (рис. 4.12).

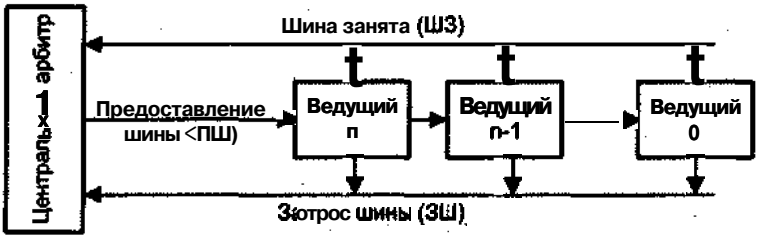


Рис. 4.12. Централизованный последовательный арбитраж с цепочкой для сигнала предоставления шины

Запросы от ведущих объединяются на линии запроса шины по схеме «монтажного ИЛИ». Аналогично организована и линия, сигнализирующая о том, что шина в данный момент занята одним из ведущих. Когда один или несколько ведущих

выставляют запросы, эти запросы транслируются на вход центрального арбитра. Получив сигнал ЗШ, арбитр анализирует состояние линии занятия шины, и если шина свободна, формирует сигнал ПШ Сигнал предоставления шины последовательно переходит по цепочке от одного ведущего к другому. Если устройство, на которое поступил сигнал ПШ, не запрашивало шину, оно просто пропускает сигнал дальше по цепочке. Когда ПШ достигнет самого левого из запросивших ведущих, последний блокирует дальнейшее распространение сигнала ПШ по цепочке и берет на себя управление шиной.

Еще раз отметим, что очередной ведущий не может приступить к управлению шиной до момента ее освобождения. Центральный арбитр не должен формировать сигнал ПШ вплоть до этого момента.

Цепочечная реализация предполагает статическое распределение приоритетов. Наивысший приоритет имеет ближайшее к арбитру ведущее устройство (устройство, на которое арбитр выдает сигнал ПШ). Далее приоритеты ведущих в цепочке последовательно понижаются.

Основное достоинство цепочечного арбитража заключается в простоте реализации и в малом количестве используемых линий. Последовательные схемы арбитража позволяют легко наращивать число устройств, подключаемых к шине.

Схеме тем не менее присущи существенные недостатки. Прежде всего, последовательное прохождение сигнала по цепочке замедляет арбитраж, причем время арбитража растет пропорционально длине цепочки. Статическое распределение приоритетов может привести к полному блокированию устройств с низким уровнем приоритета (расположенных в конце цепочки). Наконец, как и параллельный вариант, централизованный последовательный арбитраж не очень удобен в плане диагностики работы шины.

Децентрализованный арбитраж

При *децентрализованном* или *распределенном арбитраже* единый арбитр отсутствует. Вместо этого каждый ведущий содержит блок управления доступом к шине, и при совместном использовании шины такие блоки взаимодействуют друг с другом, разделяя между собой ответственность за доступ к шине. По сравнению с централизованной схемой децентрализованный арбитраж менее чувствителен к отказам претендующих на шину устройств.

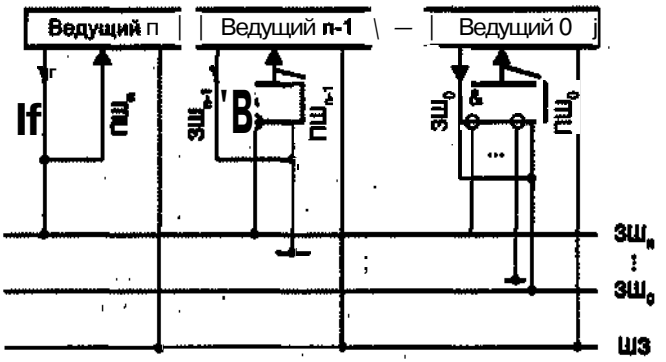


Рис. 4.13. Схема децентрализованного параллельного арбитража

Одна из возможных схем, которую можно условно назвать схемой децентрализованного параллельного арбитража, показана на рис. 4.13. Каждый ведущий имеет уникальный уровень приоритета и обладает собственным контроллером шины, способным формировать сигналы предоставления и занятия шины. Сигналы запроса от любого ведущего поступают на входы всех остальных ведущих. Логика арбитража реализуется в контроллере шины каждого ведущего.

Под децентрализованный арбитраж может быть модифицирована также схема, приведенная на рис. 4.12. Подобный вариант, называемый кольцевой схемой, показан на рис. 4.14.

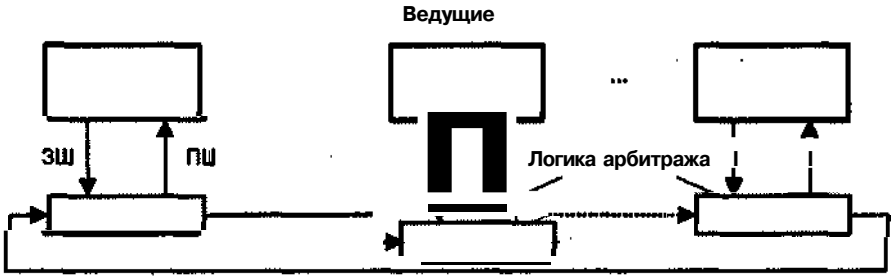


Рис. 4.14. Кольцевая схема

Здесь сигнал может возникать в различных точках цепочки, замкнутой в кольцо. Переход к новому ведущему сопровождается циклической сменой приоритетов. В следующем цикле арбитража текущий ведущий будет иметь самый низкий уровень приоритета. Соседний ведущий справа получает наивысший приоритет, а далее каждому устройству в кольце присваивается уровень приоритета на единицу меньше, чем у соседа слева. Иными словами, реализуется циклическая смена приоритетов с учетом последнего запроса.

Текущий ведущий, управляющий шиной, генерирует сигнал ПШ , который проходит через все ведущие устройства, не запросившие шину. Ведущий, сформировавший запрос и имеющий на входе активный сигнал ПШ, запрещает прохождение этого сигнала далее по цепочке, но не может взять на себя управление шиной до момента ее освобождения текущим ведущим. Когда текущий ведущий обнаруживает, что «потерял» сигнал ПШ на своем входе, он обязан при первой возможности освободить шину и снять сигнал занятия шины.

Для большинства шин все-таки более характерна другая организация децентрализованного арбитража. Такие схемы предполагают наличие в составе шины группы арбитражных линий, организованных по схеме «монтажного ИЛИ». Это позволяет любому ведущему видеть сигналы, выставленные остальными устройствами. Каждому ведущему присваивается уникальный номер, совпадающий с кодом уровня приоритета данного ведущего. Запрашивающие шину устройства выдают на арбитражные линии свой номер. Каждый из запросивших ведущих, обнаружив на арбитражных линиях номер устройствах более высоким приоритетом, снимает с этих линий младшие биты своего номера. В конце концов на арбитражных линиях остается только номер устройства, обладающего наиболее высоким приоритетом. Победителем в процедуре арбитража становится ведущий, опознав-

ший на арбитражных линиях свой номер. Подобная схема известна также *как распределенный арбитраж с самостоятельным выбором,* поскольку ведущий сам определяет, стал ли он победителем в арбитраже, то есть выбирает себя самостоятельно.

Идея подобного арбитража была предложена М. Таубом (Matthew Taub) в 1975 году. В алгоритме Тауба под арбитраж выделяются две группы сигнальных линий, доступные всем устройствам на шине. Устройства подключаются к этим линиям по схеме «монтажного ИЛИ». Первая группа служит для передачи сигналов синхронизации и управления. Вторую группу линий условно назовем шиной приоритета и обозначим В. В зависимости от принятого числа уровней приоритета эта группа может содержать от 4 до 7 линий. Каждому потенциальному ведущему назначается уникальный уровень приоритета. Приоритет Р представлен *k*-разрядным двоичным кодом. Каждому разряду кода приоритета соответствует линия в шине В. Ведущие, претендующие на управление шиной, выдают на шину В свои коды приоритета Р. Дальнейшее поведение ведущих определяется следующим правилом: если i-й разряд кода приоритета равен О ($P_i = 0$), а на i-й линии шины В в данный момент присутствует единица ($B_i = 1$), то ведущий обнуляет в выставляемом коде все младшие разряды, от 0-го до *i*-го. В результате такой процедуры на шине В остается код наивысшего из выставленных приоритетов. Устройство, распознавшее на шине свой код приоритета, считается выигравшим арбитраж. После завершения своей транзакции выигравшее устройство снимает с шины В свой код приоритета, при этом ситуация на линиях В меняется. Ведущие, претендовавшие на шину, восстанавливают ранее обнуленные разряды, и начинается новый цикл арбитража.

Поясним алгоритм примером. Пусть в некоторый момент времени запрос на шину выставили три ведущих с номерами 10,8 и 5. Положим, что уровни приоритета ведущих совпадают с их номерами, то есть на четырехразрядную шину В будут выданы соответственно коды 1010_2 (10_{10}), 1000_2 (8_{10}) и 0101_2 (5_{10}). Так как над одноименными разрядами кодов выполняется операция логического сложения (устройства подключены к линиям по схеме «монтажного ИЛИ»), на шине В установится код 1111_2. Согласно рассмотренному выше правилу первое и второе устройства обязаны обнулить разряды с 2-го по 0-й (0-й разряд - младший), а третье — все разряды. В итоге на шине В установится новый код 1000_2, и первое устройство сможет немедленно восстановить ранее обнуленные разряды. Таким образом, на шине В будет код 1010_2, то есть код приоритета устройства с номером 10. Схема арбитража этого устройства опознает на шине свой уровень приоритета и захватит шину. После завершения транзакции устройство с номером 10 снимет свой код приоритета, а остальные два устройства восстановят свои. Далее начнется новая процедура арбитража.

Описанная процедура связана с определенными затратами времени на стабилизацию на арбитражных линиях номера устройства-победителя. Это время должно быть учтено в протоколе шины.

Чтобы исключить постоянное блокирование ведущих, обладающих низким приоритетом, Тауб впоследствии модернизировал свою схему, дополнив ее модулем равнодоступности (fairness module). Модуль запрещает выигравшему ведущему выдавать новые запросы до завершения обслуживания всех ожидающих запро-

сов. Следует отметить, что это не гарантирует, а только помогает каждому ведущему получить право на управление шиной.

Вариации рассмотренной схемы широко используются в таких шинах, как Futurebus, NuBus, MultiBus II, Fastbus.

В целом схемы децентрализованного арбитража потенциально более надежны, поскольку отказ контроллера шины в одном из ведущих не нарушает работу с шиной на общем уровне. Тем не менее должны быть предусмотрены средства для обнаружения неисправных контроллеров, например на основе тайм-аута. Основной недостаток децентрализованных схем — в относительной сложности логики арбитража, которая должна быть реализована в аппаратуре каждого ведущего.

В некоторых ВМ применяют комбинированные последовательно-параллельные схемы арбитража, в какой-то мере сочетающие достоинства обоих методов. Здесь все ведущие разбиваются на группы. Арбитраж внутри группы ведется по последовательной схеме, а между группами - по параллельной.

Ограничение времени управления шиной

Вне зависимости от принятой модели арбитража должна быть также продумана стратегия ограничения времени контроля над шиной. Одним из вариантов может быть разрешение ведущему занимать шину в течение одного цикла шины, с предоставлением ему возможности конкуренции за шину в последующих циклах. Другим вариантом является принудительный захват контроля над шиной устройством с более высоким уровнем приоритета, при сохранении восприимчивости текущего ведущего к запросам на освобождение шины от устройств с меньшим уровнем приоритета.

Опросные схемы арбитража

В опросных методах запросы только фиксируются, и контроллер шины способен узнать о них, лишь опросив ведущих. Опрос может быть как централизованным — с одним контроллером, производящим опрос, так и децентрализованным — с несколькими контроллерами шины.

Данный механизм использует специальные линии опроса между контроллером (контроллерами) шины и ведущими — по одной линии для каждого ведущего. С целью уменьшения числа таких линий может формироваться номер запрашивающего ведущего, для чего вместо 2" достаточно *n* линий. Кроме того, используются также линии запроса шины и линия сигнала занятия шины.

Централизованный опрос

Централизованный опрос Иллюстрирует рис. 4.15.

Контроллер шины последовательно опрашивает каждое ведущее устройство на предмет, находится ли оно в ожидании предоставления шины. Для этого контроллер выставляет на линии опроса адрес соответствующего ведущего. Если в момент выставления адреса ведущий ожидает разрешения на управление шиной, то он, распознав свой адрес, сигнализирует об этом, делая активной шину (ЗШ). Обнаружив сигнал, контроллер разрешает ведущему использовать шину. Последовательность опроса ведущих может быть организована в порядке убывания адресов, либо меняться в соответствии с алгоритмом динамического приоритета.

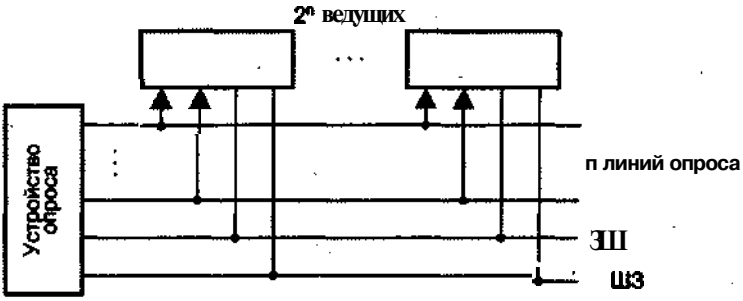


Рис. 4.15. Организация централизованного опроса ведущих

Децентрализованный опрос

Организация децентрализованного опроса показана на рис. 4.16.

Каждый ведущий содержит контроллер шины, состоящий из дешифратора адреса и генератора адреса. В начале опросной последовательности формируется адрес, который распознается контроллером. Если соответствующий ведущий ожидает доступа к шине, он вправе теперь ее занять. По завершении работы с шиной контроллер текущего ведущего генерирует адрес следующего ведущего, и процесс повторяется. При такой схеме обычно требуется применять систему с квитированием, использующую сигнал ЗШ, формируемый генератором адреса, и сигнал ПШ, генерируемый дешифратором адреса.

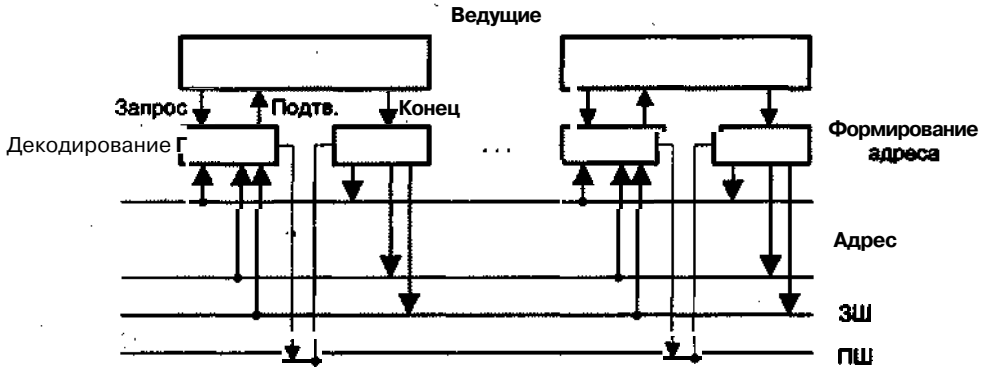


Рис. 4.16. Организация децентрализованного опроса ведущих

При децентрализованном опросе отказ в одной из точек приводит к отказу всей системы арбитража. Такая ситуация, впрочем, может быть предотвращена с помощью механизма тайм-аута: по истечении заданного времени функции отказавшего контроллера берет на себя следующий контроллер.

Протокол шины

Выставляя на шину адрес, ведущее устройство все его биты выдает на линии параллельно, что совсем не гарантирует их одновременного поступления к ведомому, устройству. Отдельные биты адреса могут преодолевать более длинный путь, дру-

гие — предварительно должны пройти через аппаратуру преобразования адресов процессора в адреса шины. Кроме того, отличия есть и в характеристиках отдельных сигнальных линий, драйверов и приемников. Рассмотренная ситуация, как уже отмечалось, называется перекосом сигналов. Прежде чем реагировать на поступивший адрес, все ведомые должны знать, с какого момента его можно считать достоверным.

Ситуация с передачей данных еще сложнее, так как данные могут пересылаться в обоих направлениях. В транзакции чтения имеет место задержка на время, пока ведомое устройство ищет затребованные данные, и ведомый должен каким-то образом известить о моменте, когда данные можно считать достоверными. Система должна предусматривать возможный перекос данных.

Метод, выбираемый проектировщиками шин для информирования о достоверности адреса, данных, управляющей информации и информации состояния, называется *протоколом шины.* Используются два основных класса протоколов — синхронный и асинхронный. В *синхронном протоколе* все сигналы «привязаны» к импульсам единого генератора тактовых импульсов (ГТИ). В *асинхронном протоколе* для каждой группы линий шины формируется свой сигнал подтверждения достоверности. Хотя в каждом из протоколов можно найти как синхронные, так и асинхронные аспекты, различия все же весьма существенны.

Синхронный протокол

В синхронных шинах имеется центральный генератор тактовых импульсов (ГТИ), к импульсам которого «привязаны» все события на шине. Тактовые импульсы (ТИ) распространяются по специальной сигнальной линии и представляют собой регулярную последовательность чередующихся единиц и нулей. Один период такой последовательности называется *тактовым периодом шины.* Именно он определяет минимальный квант времени на шине (временной слот). Все подключенные к шине устройства могут считывать состояние тактовой линии, и все события на шине отсчитываются от начала тактового периода. Изменение управляющих сигналов на шине обычно совпадает с передним или задним фронтом тактового импульса, иными словами, момент смены состояния на синхронной шине известен заранее и определяется тактовыми импульсами.

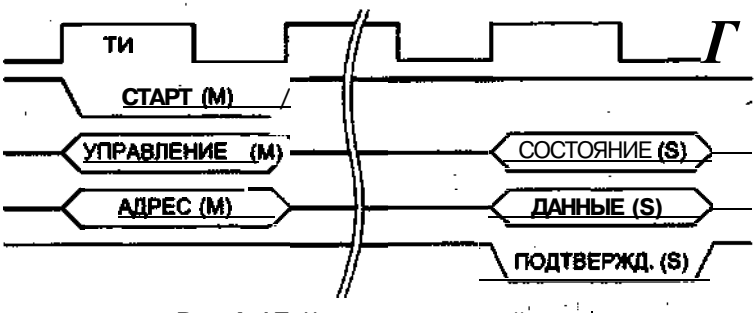


Рис. 4.17. Чтение на синхронной шине

На рис. 4.17 показана транзакция чтения с использованием простого синхронного протокола шины NuBus, применяемой в персональных компьютерах Macintosh

(буквой «M» обозначены сигналы ведущего, а буквой «S» - ведомого). Моменты изменения сигналов на шине определяет нарастающий фронт тактового импульса. Задний фронт ТИ служит для указания момента, когда сигналы можно считать достоверными. Это не обязательное условие для любых синхронных шин — во многих шинах для указания достоверности данных просто отсчитывается определенная задержка от фронта ТИ. Рассматриваемая шина NuBus имеет две особенности: тактовые импульсы асимметричны, а для передачи адреса и данных используются одни и те же сигнальные линии.

Стартовый сигнал отмечает присутствие на линиях шины адресной или управляющей информации. Когда ведомое устройство распознает свой адрес и находит затребованные данные, оно помещает эти данные и информацию о состоянии на шину и сигнализирует об их присутствии на шине сигналом подтверждения.

Операция записи выглядит сходно. Отличие состоит в том, что данные выдаются ведущим в тактовом периоде, следующем за периодом выставления адреса, и остаются на шине до отправки ведомым сигнала подтверждения и информации состояния.

Отметим, что в каждой транзакции присутствуют элементы чтения и записи, и для каждого направления пересылки имеется свой сигнал подтверждения достоверности информации на шине. Сигналы управления и адрес всегда перемещаются от ведущего. Информация состояния всегда поступает от ведомого. Данные могут перемещаться в обоих направлениях.

Хотя скорость распространения сигналов в синхронном протоколе явно не фигурирует, она должна учитываться при проектировании шины. ТИ обычно распространяются вдоль шины с обычной скоростью прохождения сигналов, и за счет определенных усилий и затрат можно добиться практически одновременной доставки ТИ к каждому разъему шины. Выбираться тактовая частота должна таким образом, чтобы сигнал от любой точки на шине мог достичь любой другой точки несколько раньше, чем завершится тактовый период, то есть шина должна допускать расхождение в моментах поступления тактовых импульсов. Ясно, что более короткие шины могут быть спроектированы на более высокую тактовую частоту.

Синхронные протоколы требуют меньше сигнальных линий, проще для понимания, реализации и тестирования. Поскольку для реализации синхронного протокола практически не требуется дополнительной логики, эти шины могут быть быстрыми и дешевыми. С другой стороны, они менее гибки, поскольку привязаны к конкретной максимальной тактовой частоте и, следовательно, к конкретному уровню технологии. По этой причине существующие шины часто не в состоянии реализовать потенциал производительности подключаемых к себе новых устройств. Кроме того, из-за проблемы перекоса синхросигналов синхронные шины не могут быть длинными.

По синхронному протоколу обычно работают шины «процессор-память».

Асинхронный протокол

Синхронная передача быстра, но в ряде ситуаций не подходит для использования. В частности, в синхронном протоколе ведущий не знает, корректно ли ответил ведомый, — возможно, он был не в состоянии удовлетворить запрос на нужные дан-

ные. Более того, ведущий должен работать со скоростью самого медленного из участвующих в пересылке данных ведомых. Обе проблемы успешно решаются в асинхронном протоколе шины.

В асинхронном протоколе начало очередного события на шине определяется не тактовым импульсом, а предшествующим событием и следует непосредственно за этим событием. Каждая совокупность сигналов, помещаемых на шину, сопровождается соответствующим синхронизирующим сигналом, называемым *стробом*. Синхросигналы, формируемые ведомым, часто называют *квитирующими сигналами* (handshakes) или *подтверждениями сообщения* (acknowledges).

На рис. 4.18 показана асинхронная операция чтения с использованием протокола, применяемого в шине Fastbus.

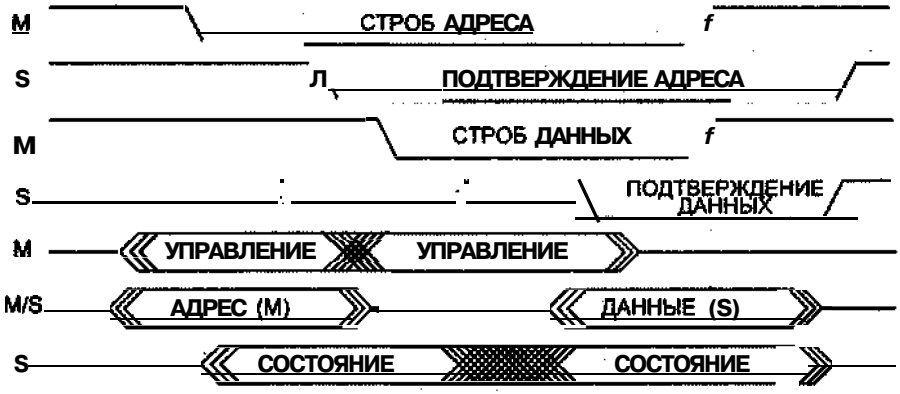


Рис. 4.18. Чтение на асинхронной шине

Сначала ведущее устройство выставляет на шину адрес и управляющие сигналы и выжидает время перекоса сигналов, после чего выдает строб адреса, подтверждающий достоверность информации. Ведомые следят за адресной шиной, чтобы определить, должны ли они реагировать. Ведомый, распознавший на адресной шине свой адрес, отвечает информацией состояния, сопровождаемой сигналом подтверждения адреса. Когда ведущий обнаруживает подтверждение адреса, он знает, что соединение установлено, и готов к анализу информации состояния. Присутствие адреса на ША далее не требуется, поскольку у ведомого уже имеется копия полного адреса либо нужной его части.

Далее ведущий меняет управляющую информацию, выжидает время перекоса и выдает строб данных. Если это происходит в транзакции записи, то ведущий одновременно с управляющей информацией выставляет на шину записываемые данные. В рассматриваемом случае управляющая информация извещает ведомое устройство, что это чтение. Когда ведомый подготовит затребованные данные, он выдает их на шину совместно с новой информацией о состоянии и формирует сигнал подтверждения данных. Когда ведущий видит сигнал подтверждения данных, он читает данные с шины и снимает строб данных, чтобы показать, что действия с данными завершены. В нашем примере ведущий снимает также и строб адреса. В более сложных вариантах транзакций строб адреса может оставаться на шине для поддержания соединения в течение нескольких циклов данных. Обнаружив

исчезновение строба данных, ведомый снимает с шины данные и информацию состояния, а также сигнал подтверждения данных, переводя шину в свободное состояние.

В цикле асинхронной шины для подтверждения успешности транзакции используется двунаправленный обмен сигналами управления. Такая процедура носит название *квитирования установления связи* или *рукопожатия* (handshake). В рассмотренном варианте процедуры ни один шаг в передаче данных не может начаться, пока не завершен предыдущий шаг. Такое квитирование известно как *квитирование с полной взаимоблокировкой* (fully interlocked handshake).

Как и в синхронных протоколах, в любой асинхронной транзакции присутствуют элементы чтения и записи: по отношению к управляющей информации выполняется операция записи, а к информации состояния - чтения. Данные синхронизируются и управляются, соответственно, как управляющая и статусная информация.

Скорость асинхронной пересылки данных диктуется ведомым, поскольку ведущему для продолжения транзакции приходится ждать отклика. Асинхронные протоколы по своей сути являются самосинхронизирующимися, поэтому шину могут совместно использовать устройства с различным быстродействием, построенные на базе как старых, так и новых технологий. Шина автоматически адаптируется к требованиям устройств, обменивающихся информацией в данный момент. Таким образом, с развитием технологий к шине могут быть подсоединены более быстрые устройства, и пользователь сразу ощутит всё их преимущества. В отличие от синхронных систем для ускорения системы с асинхронной шиной не требуется замена на шине старых медленных устройств на быстрые новые. Платой за перечисленные преимущества асинхронного протокола служит некоторое увеличение сложности аппаратуры.

Квитирование в асинхронных системах не всегда реализуется в полном объеме. Иногда транзакция на шине не может быть завершена стандартным образом, например, если ведущий из-за программных ошибок обращается к несуществующей ячейке памяти. В этом случае ведомое устройство не отвечает соответствующим подтверждающим сигналом. Чтобы предотвратить бесконечное, ожидание в шинах, используется тайм-аут, то есть задается время, спустя которое при отсутствии отклика транзакция принудительно прекращается. Для реализации тайм-аута необходимы схемы, способные решать, пришло ли подтверждение вовремя, и если нет, то как привести шину к исходному состоянию. Первая часть решается с помощью таймера, запускаемого ведущим одновременно с началом транзакции. Если таймер достигает предопределенного значения до поступления ответного сигнала, ведущий обязан прекратить начатую транзакцию. Восстановление состояния шины и вычислительного процесса после тайм-аута может происходить по-разному. Так, если ведущим устройством является процессор, он делает это с помощью специального вызова операционной системы, известного как «ошибка шины» (bus error).

Тайм-ауты цикла данных, обычно означающие отказ оборудования, достаточно редки, поэтому время тайм-аута может быть весьма большим. С другой стороны, тайм-ауты по адресу возникают часто. Происходит это, например, когда программа инициализирует систему и проверяет, какие из устройств присутствуют на шине; при этом вполне реальна выдача на шину адреса несуществующего уст-

ройства. В спецификациях шин предписываются очень малые значения тайм-ута по адресу, из-за чего устройства декодирования адреса в ведомых устройствах должны быть весьма быстрыми, чтобы уложиться в отведенное время.

Шины ввода/вывода обычно реализуются как асинхронные.

Особенности синхронного и асинхронного протоколов

В предшествующие годы проектировщики ВМ отдавали предпочтение асинхронным шинам, однако в последних высокопроизводительных разработках все чаще используются шины на базе синхронного протокола,

Существующие синхронные шины несколько быстрее асинхронных и по уровню быстродействия уже достаточно близки к физическим скоростям распространения сигналов. Возможное ускорение асинхронных шин по мере развития технологии может привести лишь к незначительному их преимуществу над синхронными.

Любой из протоколов предполагает информирование схем арбитража о занятости шины. В синхронном протоколе шина занята от начала стартового сигнала до завершения сигнала подтверждения, и специальный сигнал занятости шины не нужен. В асинхронном протоколе о занятости шины свидетельствуют адресный строб или сигнал подтверждения адреса.

В асинхронной системе присутствует полная процедура квитирования установления связи, то есть во всех случаях оба устройства до удаления информации с шины должны прийти к соглашению. Таким образом, даже если одно из них построено на очень быстрых схемах, а другое — на очень медленных, взаимодействие все равно будет успешным.

Синхронные системы квитируются частично, за счет того, что ведомое устройство перед выдачей подтверждения может занимать под поиск нужных данных несколько тактовых периодов. С другой стороны, существует неявное требование, чтобы ведомый успел использовать или, по крайней мере, скопировать адрес и информацию управления за время одного тактового периода, до их исчезновения с сигнальных линий. Необходимо также, чтобы и считывание данных ведущим также происходило а пределах одного тактового периода, иначе эти данные будут утеряны. Если в ведущем устройстве используется динамическая память, это требование может приводить к проблемам, если в момент получения данных память находится в режиме регенерации. Для решения подобных проблем обычно используют дополнительную буферную память. Отметим также, что если ведомому для завершения своей операции требуется время, лишь незначительно превышающее длительность тактового периода, транзакция все равно удлиняется на целый период. Это существенный недостаток по сравнению с асинхронным протоколом.

В обоих видах протоколов необходимо учитывать эффект перекоса сигналов. Максимальное значение времени перекоса равно разности времен прохождения сигналов по самой быстрой и самой медленной сигнальным линиям шины. В синхронных шинах перекос уже заложен в указанную в спецификации максимальную тактовую частоту, поэтому при проектировании устройств может не учитываться. Для асинхронных шин перекос необходимо принимать во внимание для каждой транзакции и для каждого устройства. Перед выставлением строба веду-

щее устройство выжидает в течение времени перекоса данных, считая от момента выставления на шину данных, так, что когда ведомый видит строб, он уже может считать данные достоверными. У ведомого дополнительно возможен перекос сигналов на внутренних трактах данных. Компенсировать его можно введением принудительной задержки, перед тем как использовать полученный сигнал стробирования. Когда ведомый возвращает данные ведущему, он должен после установки данных на шине, но до отправки сигнала подтверждения выждать время перекоса.

Учет перекоса может быть реализован как в ведущем, так и в ведомом устройстве, либо и там и там, лишь бы была обеспечена необходимая общая задержка. Например, в шине Fastbus, где количество ведущих обычно много меньше числа ведомых, ответственность за учет перекоса сигналов возлагается на ведущих, благодаря чему сокращается число устройств, которые потребуют модификации при изменении свойств шины. В рассматриваемой шине ведомый выставляет подтверждение одновременно с данными, а ведущий перед считыванием данных выжидает в течение времени перекоса. Величина компенсирующей задержки зависит от технологии шины, а также физических свойств и длины ее сигнальных линий. В свою очередь, ведомые должны самостоятельно отвечать за проблемы, связанные с их внутренними перекосами сигналов.

В обоих протоколах необходимо учитывать еще одну проблему - *проблему метастабильного состояния*. Суть ее поясним на примере микропроцессора, к которому подключена клавиатура. Время от времени микропроцессор считывает информацию из регистра состояния клавиатуры, который должен «решить», была ли нажата клавиша, и в зависимости от этого возвратить единицу или ноль. Проблема возникает, если принятие решения практически совпадает с моментом опроса регистра. Если это происходит несколько раньше, регистр вернет 1, а если чуть позже, то 0, но факт нажатия запоминается в соответствующем триггере регистра состояния и будет зафиксирован при следующем опросе регистра. Сложность заключается в том, что в момент переключения триггера информация на входе должна оставаться неизменной. В спецификации на реальные триггеры указывается интервал вблизи тактового импульса, в течение которого входная информация не должна изменяться. Если данные не синхронизированы с ТИ и поступают от какого-либо независимого источника, как в примере с клавиатурой, предотвратить изменение входной информации триггера в запрещенном интервале невозможно. При нарушении данного условия триггер способен перейти в метастабильное состояние, то есть на его выходе может на неопределенное время установиться неоднозначный уровень напряжения, который сохранится, пока случайный шум не установит триггер в то или иное стабильное состояние.

Метастабильное состояние триггера опасно неопределенным поведением схем, для которых информация триггера является входной. К сожалению, многие проектировщики игнорируют упомянутую проблему, и это становится причиной случайных ошибок.

Кардинально решить означенную проблему принципиально невозможно, поэтому при проектировании необходимо проявить особую тщательность, чтобы уменьшить вероятность возникновения метастабильного состояния. Одним из методов может быть правильный выбор элементов, поскольку некоторые триггеры срабатывают быстрее, чем иные. Эффективными способами могут являться

использование двухтактных триггеров и/или синхронизация триггеров тактовыми импульсами, что снижает вероятность ошибки до уровня несущественной.

В асинхронных системах имеется иная возможность: специальные схемы для обнаружения метастабильных состояний, где асинхронная система вправе просто выждать, пока состояние не станет стабильным.

Методы повышения эффективности шин

Существует несколько приемов, позволяющих повысить производительность шин. К ним, прежде всего, следует отнести пакетный режим, конвейеризацию и расщепление транзакций.

Пакетный режим пересылки информации

Эффективность как выделенных, так и мультиплексируемых шин может быть улучшена, если они функционируют в *блочном* или *пакетном режиме* (burst mode), когда один адресный цикл сопровождается множественными циклами данных (чтения или записи, но не чередующимися). Это означает, что пакет данных передается без указания текущего адреса внутри пакета.

При записи в память последовательные элементы блока данных заносятся в последовательные ячейки. Так как в пакетном режиме передается адрес только первой ячейки, все последующие адреса генерируются уже в самой памяти путем последовательного увеличения начального адреса Передача на устройства ввода/вывода или в память наподобие очереди может не сопровождаться изменением начального адреса.

Скорость передачи собственно данных в пакетном режиме увеличивается естественным образом за счет уменьшения числа передаваемых адресов. Внутри пакета очередные данные могут передаваться в каждом такте шины, длина пакета может достигать 1024 байт. Наиболее частый вариант - пакеты, состоящие из четырех байтов. Такие пакеты используются при работе с памятью в 32-разрядных ВМ, где длина ячейки памяти равна одному байту.

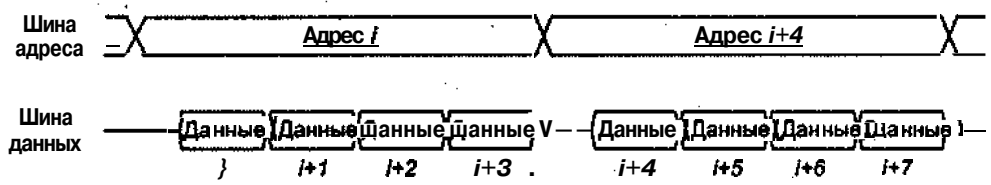


Рис. 4.19. Пакетный режим передачи данных -

Рисунок 4.19 иллюстрирует концепцию адресации в пакетном режиме при пересылке данных. По шине адреса передается только адрес ячейки *i,* а в данных для ячеек $i + 1$, $i + 2$ и $i + 3$ указание соответствующих адресов отсутствует.

В асинхронных системах пакетный режим позволяет достичь дополнительного эффекта. Как известно, время пересылки слова включает в себя время прохождения слова от отправителя к приемнику и время, затрачиваемое на процедуру подтверждения. Необходимо также учесть внутренние задержки в ведущем и ведо-

мом устройствах и, наконец, дополнительные издержки на восстановление исходного состояния шины после процедуры квитирования. В ходе пакетной передачи можно избавиться от этих задержек и работать с максимальной пропускной способностью, которую допускают ширина полосы пропускания линий и перекос сигналов, за счет разрешения отправителю начинать следующий цикл данных не ожидая подтверждения. Реализация описанного режима сопряжена с некоторыми ограничениями. В частности, становится невозможным восстановление ошибок в каждом цикле. Кроме того, скорость должна быть тщательно согласована с особенностями каждой передачи.

Примером шины, обеспечивающей пакетный режим передачи, может служить современная шина Futurebus+.

Конвейеризация транзакций

Одним из способов повышения скорости передачи данных по шине является *конвейеризация транзакций*. Очередной элемент данных может быть отправлен устройством А до того, как устройство В завершит считывание предыдущего элемента. Аналогичное решение уже рассматривалось в разделе, посвященном пакетному режиму, однако сам прием применим и к обычным транзакциям.

На рис. 4.20 показана конвейеризация транзакций чтения.

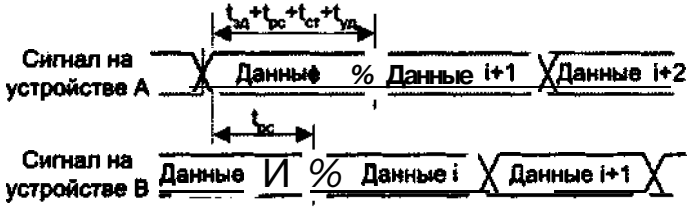


Рис. 4.20. Конвейеризация транзакций чтения

Данные на шине должны оставаться стабильными в течение времени $t_{ст} + t_{уд}$. Только после этого возможна смена элемента данных. Максимальная скорость передачи при конвейеризации определяется выражением $\frac{1}{t_{ст} + t_{уд}}$.

Протокол с расщеплением транзакций

Для увеличения эффективной полосы пропускания шины во многих современных шинах используется протокол *с расщеплением транзакций* (split transaction), известный также как *протокол соединения/разъединения* (connect/disconnect) или *протокол с коммутацией пакетов* (packet-switched). Этот протокол обычно обеспечивает преимущество на транзакциях чтения.

В классическом варианте любая транзакция на шине неразрывна, то есть новая транзакция может начаться только после завершения предыдущей, причем в течение всего периода транзакции шина остается занятой. Протокол с расщеплением транзакций допускает совмещение во времени сразу нескольких транзакций.

В шине с расщеплением транзакций линии адреса и данных обязаны быть независимыми. Каждая транзакция чтения разделяется на две части: адресную тран-

закцию и транзакцию данных. Считывание данных из памяти начинается с адресной транзакции: выставления ведущим на адресную шину адреса ячейки. С приходом адреса память приступает к относительно длительному процессу поиска и извлечения затребованных данных. По завершении чтения память становится ведущим устройством, запрашивает доступ к шине и направляет считанные данные по шине данных. Фактически от момента поступления запроса до момента формирования отклика шина остается незанятой и может быть востребована для выполнения других транзакций. В этом и состоит главная идея протокола расщепления транзакций.

Таким образом, на шине с расщеплением транзакции имеют место поток запросов и поток откликов. Часто в системах с расщеплением транзакций контроллер памяти проектируется так, чтобы обеспечить буферизацию множественных запросов.

Случай, когда затребованные данные возвращаются в той же последовательности, в которой поступали запросы, в сущности, представляет собой рассмотренную выше конвейеризацию. Шина с расщеплением транзакций зачастую может обеспечивать вариант, при котором ответы на запросы поступают в произвольной последовательности (рис. 4.21). Чтобы не спутать, какому из запросов соответствует информация на шине данных, ее необходимо снабдить признаком (тегом).

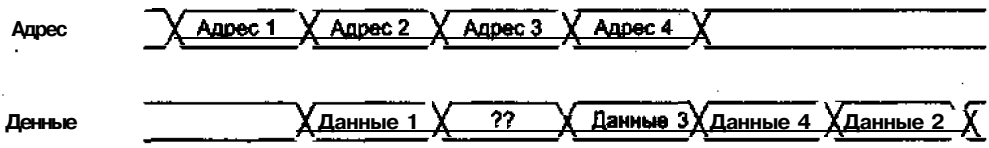


Рис. 4. 2 1. Расщепление транзакций

Хотя протокол с расщеплением транзакций и позволяет более эффективно использовать полосу пропускания шины по сравнению с протоколами, удерживающими шину в течение всей транзакции, он обычно вносит дополнительную задержку из-за необходимости получать два подтверждения — при запросе и при отклике. Кроме того, реализация протокола связана с дополнительными затратами, так как требует, чтобы транзакции были тегированы и отслеживались каждым устройством.

- Для любой шины с расщеплением транзакций существует предельное значение числа одновременно обслуживаемых запросов.

Увеличение полосы пропускания шины

Среди приемов, способствующих расширению полосы пропускания шины, основными, пожалуй, можно считать следующие:

- отказ от мультиплексирования шин адреса и данных;
- увеличение ширины шины данных;
- повышение тактовой частоты шины;
- использование блочных транзакций.

Замена мультиплексируемой шины адреса/данных и переход к выделенным шинам адреса и данных делают возможной одновременную пересылку как адреса, так и данных, то есть позволяют реализовать более эффективные варианты тран-

закций. Такое решение, однако, является более дорогостоящим из-за необходимости иметь большее число сигнальных линий.

Полоса пропускания шины по своему определению непосредственно зависит от количества параллельно пересылаемой информации — практически прямо пропорциональна ширине шины данных. Несмотря на то что данный способ требует увеличения числа сигнальных линий, многие разработчики ВМ используют в своих машинах достаточно широкие шины данных. Например, в рабочей станции SPARCstation 20 ширина шины составляет 128 бит.

Наращивание тактовой частоты - еще один очевидный способ увеличения полосы пропускания, и проектировщики широко им пользуются.

О том, как на полосу пропускания шины влияют пакетные или блочные транзакции, было сказано выше: Данный способ требует некоторого усложнения аппаратуры, но одновременно позволяет сократить время обслуживания запроса.

Ускорение транзакций

Для сокращения времени транзакций проектировщики обычно прибегают к следующим приемам:

- арбитражу с перекрытием;
- арбитражу с удержанием шины;
- расщеплению транзакций.

Сущность расщепления транзакций была рассмотрена ранее. Кратко поясним остальные два метода.

Арбитраж с перекрытием (overlapped arbitration) заключается в том, что одновременно с выполнением текущей транзакции производится арбитраж следующей транзакции.

При *арбитраже с удержанием шины* (bus parking) ведущий может удерживать шину и выполнять множество транзакций, пока отсутствуют запросы от других потенциальных ведущих.

В современных шинах обычно сочетаются все вышеперечисленные способы ускорения транзакций.

Повышение эффективности шин с множеством ведущих

Любая система шин характеризуется пределом пропускной способности, зависящим от их ширины, скорости и протокола. Имеются также издержки, такие как арбитраж, если только он не проводится параллельно с выполнением предшествующей транзакции. Даже простой микропроцессор способен практически монополизировать производительность объединительной шины при выборке инструкций и данных, но без блочных пересылок.

При проектировании мультипроцессорных систем целесообразно рассматривать системную шину как коммуникационный тракт между разными процессорами и нескольким контроллерами ввода/вывода и снабдить каждый процессор локальной памятью для команд и большей части данных. Это существенно снижает нагрузку на системную шину. Если процессоры используют шину в первую очередь для ввода/вывода и пересылки сообщений, большая часть трафика может быть

реализована в виде блочных пересылок, что ведет практически к удвоению пропускной способности. Однако, в зависимости от числа процессоров и природы приложения, шина может стать и «узким местом». Фактически, если шина в течение значительной части времени не свободна, процессоры могут значительную долю времени провести в состоянии ожидания. Система с пересылкой сообщений начинает функционировать скорее как сеть, чем как простая шина ввода/вывода.

Одно из решений проблемы пропускной способности - увеличение количества шин с несколькими процессорами на каждой. Этот подход применен в шине Fastbus, где общее адресное пространство совместно используется нескольким отдельными шинами, называемыми сегментами. Сегменты функционируют независимо, но автоматически объединяются нужным образом, если ведущий из одного сегмента обращается к ведомому из другого сегмента. Это автоматическое объединение выражается во вмешательстве в трафик всех промежуточных сегментов, поэтому, чтобы не возникало заторов, применяться оно должно осторожно. Разумное использование узлов с промежуточным хранением совместно с сетевым протоколом передачи сообщений могут еще более сократить перегрузку путем сглаживания нагрузки, разрешая одновременное объединение как двух, так и нескольких сегментов.

Надежность и отказоустойчивость

Надежность и отказоустойчивость - важнейшие аспекты проектирования шин. Основные надежды здесь обычно возлагают на корректирующие коды, которые позволяют обнаружить отказ одиночного элемента или шумовой выброс и автоматически парировать ошибку. Подобный подход, широко практикуемый в системах памяти, применительно к шинам порождает специфические проблемы.

В шинах отдельные функциональные группы сигналов (сигналы адреса, данных, управления, состояния и арбитража) предполагают независимые контроль и коррекцию. При наличии множества небольших групп сигналов метод корректирующих кодов становится малоэффективным из-за необходимости включения в шину большого числа контрольных линий. Кроме того, в шинах весьма вероятно одновременное возникновение ошибок сразу в нескольких сигналах. Для учета такой ситуации необходимо увеличивать разрядность корректирующего кода, то есть вводить в шину дополнительные сигнальные линии. Достаточно неясным остается вопрос защиты одиночных сигналов, в частности сигналов тактирования и синхронизации.

Вычисление корректирующих кодов и коррекция ошибок требуют дополнительного времени, что замедляет шину.

Усложнение аппаратуры, обусловленное использованием корректирующих кодов, ведет к снижению общей надежности шины, в результате чего суммарный выигрыш может оказаться меньше ожидаемого. В силу приведенных соображений становится ясным, почему проектировщики постоянно ищут альтернативные способы обеспечения надежности и отказоустойчивости шин.

Достаточно хорошие результаты дают так называемые «высокоуровневые» подходы. Здесь вместо отслеживания каждого цикла шины производятся контроль и коррекция более крупных единиц, например целых блоков данных или законченной программной операции.

При наличии в системе избыточных процессоров и шин возможен перекрестный контроль, причем программное обеспечение может производить изменения в конфигурации системы и предупреждать оператора о необходимости замены определенных блоков. Даже если шина имеет встроенные средства коррекции ошибок, желательно дополнять их некоторым дополнительным уровнем «разумности» для предотвращения такой постепенной деградации системы, компенсировать которую имеющийся механизм коррекции будет уже не в состоянии.

При разработке аппаратуры необходимо обязательно учитывать определенные требования, связанные с обеспечением отказоустойчивости. Так, если обнаружена ошибка, то для ее коррекции должна быть предусмотрена возможность повторной передачи данных. Это предполагает, что оригинальная передача не должна приводить к необратимым побочным эффектам. Например, если операция чтения с периферийного устройства вызывает стирание исходных данных или сбрасывает флаги состояния, успешное повторное чтение становится невозможным. Другой пример: работа с буферной памятью типа FIFO (First In First Out), работающей по принципу «первым прибыл, первым обслужен», где ошибочные данные внутри очереди недоступны и поэтому не могут быть откорректированы.

Чтобы учесть подобные ситуации, при разработке адресуемой памяти необходимо предусмотреть буферы, а очистка ячеек и сброс флагов должны быть не побочными эффектами, а выполняться только явно с помощью определенных команд. Память типа FIFO может быть снабжена адресуемыми буферами, предназначенными для хранения данных вплоть до завершения передачи.

Стандартизация шин

Стандартизация шин позволяет разработчикам различных устройств вычислительных машин работать независимо, а пользователям - самостоятельно сформировать нужную конфигурацию-ВМ. Появление стандартов зависит от разных обстоятельств. Часто стандарты разрабатываются специализированными организациями. Так, общепризнанными авторитетами в области стандартизации являются IEEE (Institute of Electrical and Electronics Engineers) - Институт инженеров по электротехнике и электронике) и ANSI (American National Standards Institute) - Национальный институт стандартизации США. Многие стандарты становятся итогом кооперации усилий производителей оборудования для вычислительных машин. Иногда в силу популярности конкретных машин реализованные в них решения становятся стандартами де-факто, однако успех таких стандартов во многом определяется их принятием и утверждением в IEEE и ANSI.

В табл. 4.2-4.5 приведены основные характеристики некоторых распространенных шин, как стандартных, так и претендующих на роль таковых.

Таблица 4.2. Стандартные системные шины общего применения

Характеристика	VME	Futurebus	Multibus II
Разработчик	Motorola, Philips, Mostek	IEEE	Intel
Ширина шины	128	.96	96

Характеристика	VME	Futurebus	Multibus II
Мультиплексирование адреса/данных	Нет	Да	Да
Разрядность адреса, бит	16/24/32/64	32	
Разрядность данных, бит	8/16/32/64	16/32/64/128	32
Вид пересылки	Одиночная или групповая	Одиночная или групповая	Одиночная или групповая
Количество ведущих	Несколько	Несколько	Несколько
Арбитраж	Централизованный	Централизованный или децентрализованный	Децентрализованный
Расщепление транзакций	Нет	Возможно	Возможно
Протокол	Асинхронный	Асинхронный	Синхронный
Тактовая частота, МГц	Нет данных	Нет данных	10
Полоса пропускания при одиночной пересылке, Мбайт/с	25	37	20
Полоса пропускания при групповой пересылке, Мбайт/с	28	95	40/80
Максимальное количество устройств	21	20	21
Максимальная длина шины, м	0,5	0,5	0,5
Стандарт	ШЕЕ 1014	IEEE 896.1	ANSI/IEEE 1296

Таблица 4.3. Системные шины высокопроизводительных серверов

Характеристика	Summit	Challenge	XDBus
Разработчик	HP	SGI	Sun
Мультиплексирование адреса/данных	Нет данных	Нет данных	Да
Разрядность адреса, бит	48	40	Нет данных
Разрядность данных, бит	128/512	256/1024	144/512
Вид пересылки	Одиночная или групповая	Одиночная или групповая	Одиночная или групповая
Количество ведущих	Несколько	Несколько	Несколько
Арбитраж	Централизованный	Централизованный	Централизованный
Расщепление транзакций	Есть	Есть	Есть

продолжение

Таблица 4.3 *(продолжение)*

Характеристика	Summit	Challenge	XDBus
Протокол	Синхронный	Синхронный	Синхронный
Тактовая частота, МГц	60	48	*66*
Полоса пропускания при одиночной пересылке, Мбайт/с	60	48	*66*
Полоса пропускания при групповой пересылке, Мбайт/с	.960	1200	1056
Максимальная длина шины, м	0,3	0,3	0,4
Стандарт	Нет	Нет	Нет

Таблица 4.4. Системные шины персональных вычислительных машин

Характеристика	NuBus	ISA 8/16	EISA	FSB Pentium 4
Разработчик	Texas Instruments	IBM	AST, Compaq, Epson, HP, NEC, Olivetti, Tandy, Wyse, Zenith	Intel
Ширина шины	96	62/98	98/100	Нет данных
Мультиплексирование	Да	Нет	Нет	Нет
Разрядность адреса, бит	32	20/24	24/32	36
Разрядность данных, бит	32	8/16	16/32	64/128
Вид пересылки	Одиночная или групповая	Одиночная или групповая	Одиночная или групповая	Одиночная или групповая
Арбитраж	Централизованный	Нет данных	Централизованный	Нет данных
Расщепление транзакций	Нет	Нет данных	Возможно	Да
Количество ведущих	Несколько (ограничено)	Один	Один	Нет данных
Протокол	Синхронный	Синхронный	Синхронный	Синхронный
Тактовая частота, МГц	10	4,77/8,33	8,33	400(баз.100); 533(баз.133); 800(ожидается)

Характеристика	NuBus	ISA 8/16	EISA	FSB Pentium 4
Полоса пропускания при одиночной пересылке, Мбайт/с	40	33	33	1060(133); 3200(400); 4200(533)

Таблица 4.5. Шины ввода/вывода

Характеристика	PCI	SCSI	SCSI-2	IDE
Разработчик	Intel			
Ширина шины	124/128	50	Варьируется	40
Мультиплексирование адреса/данных	Да	Да	Да	Нет
Разрядность адреса, бит	32/64	Нет данных	Нет данных	2
Разрядность данных, бит	32/64	8	8/16/32	16
Вид пересылки	Одиночная или групповая	Одиночная или групповая	Одиночная или групповая	Групповая
Количество ведущих	Несколько	Несколько	Несколько	Один
Арбитраж	Централизованный	Децентрализованный	Децентрализованный	Нет данных
Расщепление транзакций	Нет	Возможно	Возможно	Нет
Протокол	Синхронный	Синхронный и асинхронный	Синхронный и асинхронный	Асинхронный
Тактовая частота, МГц	33/66	5/10	10/20/40/80	Нет данных
Полоса пропускания при одиночной пересылке, Мбайт/с	33	1,5 (асинхронный); 5 (синхронный)	5-40 (синхронный)	Нет данных
Полоса пропускания при групповой пересылке, Мбайт/с	132/520	1,5 (асинхронный); 5(синхронный)	40/80/160/320(синхронный)	До 200
Максимальное количество устройств	Нет данных	7	7	2(только диски)
Максимальная длина шины, м	0,5	25	25	0,5
Стандарт	Нет	ANSI X3.131-1986	ANSI X3.131-199x	ANSI X3T9.2/90-14

Контрольные вопросы

1. Перечислите основные виды структур взаимосвязей вычислительной машины.
2. Какие параметры включает в себя полная характеристика шины?

3. Что такое транзакция, из каких этапов она состоит?
4. В чем заключается основное различие между ведущими и ведомыми устройствами?
5. Что такое широковещательный режим записи?
6. Какие шины в составе ВМ образуют иерархию шин?
7. Какие параметры определяют механические аспекты шины?
8. Каким образом устройства подключаются к линиям шины?
9. Что такое перекос сигналов и каковы методы компенсации этого эффекта?
10. Определите понятия «драйвер шины», «приемник», «трансивер».
11. Поясните причины возникновения переходной помехи.
12. Как в шинах организуется система заземления?
13. Что такре «широковещательный опрос» и как он реализуется?
14. Какая характеристика вычислительной машины зависит от ширины адресной шины?
15. В чем заключаются основные преимущества и недостатки мультиплексирования адреса и данных?
16. Какие группы сигнальных линий образуют шину управления?
17. Определите задачи арбитража шин.
18. Перечислите и охарактеризуйте схемы динамической смены приоритетов.
19. Сформулируйте преимущества и недостатки централизованного и децентрализованного арбитража.
20. Чем определяются приоритеты устройств при цепочечном арбитраже?
21. Поясните схему арбитража Тауба.
22. Какими способами организуются опросные виды арбитража?
23. Каким образом выполняются транзакции в шинах с синхронным протоколом?
24. Поясните последовательность действий в процедуре квитирования установления связи.
25. Какие средства обеспечивают исключение бесконечного ожидания ответа в асинхронном протоколе?
26. Для каких шин наиболее характерен пакетный режим передачи информации?
27. В чем суть и достоинства конвейеризации транзакций?
28. Какие дополнительные преимущества предоставляет расщепление транзакций?
29. Какими средствами можно увеличить полосу пропускания шины?
30. Перечислите способы ускорения транзакций на шине.
31. Какие приемы обеспечивают повышение эффективности шин с множеством ведущих?
32. Охарактеризуйте средства повышения надежности шин.

Глава 5
Память

В любой ВМ, вне зависимости от ее архитектуры, программы и данные хранятся в памяти. Функции памяти обеспечиваются запоминающими устройствами (ЗУ), предназначенными для фиксации, хранения и выдачи информации в процессе работы ВМ. Процесс фиксации информации в ЗУ называется *записью,* процесс выдачи информации — *чтением* или *считыванием,* а совместно их определяют как *процессы обращения* к ЗУ.

Характеристики систем памяти

Перечень основных характеристик, которые необходимо учитывать, рассматривая конкретный вид ЗУ, включает в себя:

- месторасположения;
- емкость;
- единицу пересылки;
- метод доступа;
- быстродействие;
- физический тип;
- физические особенности;
- стоимость.

По месту расположения ЗУ разделяют на процессорные, внутренние и внешние. Наиболее скоростные виды памяти (регистры, кэш-память первого уровня) обычно размещают на общем кристалле с центральным процессором, а регистры общего назначения вообще считаются частью ЦП. Вторую группу (внутреннюю память) образуют ЗУ, расположенные на системной плате. К внутренней памяти относят основную память, а также кэш-память второго и последующих уровней (кэш-память второго уровня может также размещаться на кристалле процессора). Медленные ЗУ большой емкости (магнитные и оптические диски, магнитные ленты) называют внешней памятью, поскольку к ядру ВМ они подключаются аналогично устройствам ввода/вывода.

Емкость ЗУ характеризуют числом битов либо байтов, которое может храниться в запоминающем устройстве. На практике применяются более крупные единицы, а для их обозначения к словам «бит» или «байт» добавляют приставки: кило, мега, гига, тера, пета, экза (kilo, mega, giga, tera, peta, exa). Стандартно эти приставки означают умножение основной единицы измерений на 10^3,10^6, 10^9, 10^{12}, 10^{15} и 10^{18} соответственно. В вычислительной технике, ориентированной на двоичную систему счисления, они соответствуют значениям достаточно близким к стандартным, но представляющим собой целую степень числа 2, то есть 2^{10}, 2^{20}, 2^{30},2^{40},2^{50}, 2^{60}. Во избежание разночтений, в последнее время ведущие международные организации по стандартизации, например IEEE (Institute of Electrical and Electronics Engineers), предлагают ввести новые обозначения, добавив к основной приставке слово binary (бинарный): kilobinary, megabinary, gigabinary, terabinary, petabinary, exabinary. В результате вместо термина «килобайт» предлагается термин «кибибайт», вместо «мегабайт» - «мебибайт» и т. д. Для обозначения новых единиц предлагаются сокращения: Ki, Mi, Gi, Ti, Pi и Ei [133].

Важной характеристикой ЗУ является *единица пересылки*. Для основной памяти (ОП) единица пересылки определяется шириной шины данных, то есть количеством битов, передаваемых по линиям шины параллельно. Обычно единица пересылки равна длине слова, но не обязательно. Применительно к внешней памяти данные часто передаются единицами, превышающими размер слова, и такие единицы называются *блоками*.

При оценке быстродействия необходимо учитывать применяемый в данном типе *ЗУ метод доступа* к данным. Различают четыре основных метода доступа:

- **Последовательный доступ.** ЗУ с последовательным доступом ориентировано на хранение информации в виде последовательности блоков данных, называемых записями. Для доступа к нужному элементу (слову или байту) необходимо прочитать все предшествующие ему данные. Время доступа зависит от положения требуемой записи в последовательности записей на носителе информации и позиции элемента внутри данной записи. Примером может служить ЗУ на магнитной ленте.
- **Прямой доступ.** Каждая запись имеет уникальный адрес, отражающий ее физическое размещение на носителе информации. Обращение осуществляется *как* адресный доступ к началу записи, с последующим последовательным доступом к определенной единице информации внутри записи. В результате время доступа к определенной позиции является величиной переменной. Такой режим характерен для магнитных дисков.
- **Произвольный доступ.** Каждая ячейка памяти имеет уникальный физический адрес. Обращение к любой ячейке занимает одно и то же время и может производиться в произвольной очередности. Примером могут служить запоминающие устройства основной памяти.
- **Ассоциативный доступ.** Этот вид доступа позволяет выполнять поиск ячеек, содержащих такую информацию, в которой значение отдельных битов совпадает с состоянием одноименных битов в заданном образце. Сравнение осуществляется параллельно для всех ячеек памяти, независимо от ее емкости. По ассоциативному принципу построены некоторые блоки кэш-памяти.

Быстродействие ЗУ является одним из важнейших его показателей. Для количественной оценки быстродействия обычно используют три параметра:

- *Время доступа* ($T_д$). Для памяти с произвольным доступом оно соответствует интервалу времени от момента поступления адреса до момента, когда данные заносятся в память или становятся доступными. В ЗУ с подвижным носителем информации - это время, затрачиваемое на установку головки записи/считывания (или носителя) в нужную позицию.
- *Длительность цикла памяти* или *период обращения* ($T_ц$). Понятие применяется к памяти с произвольным доступом, для которой оно означает минимальное время между двумя последовательными обращениями к памяти. Период обращения включает в себя время доступа плюс некоторое дополнительное время. Дополнительное время может требоваться для затухания сигналов на линиях, а в некоторых типах ЗУ, где считывание информации приводит к ее разрушению, - для восстановления считанной информации.
- *Скорость передачи.* Это скорость, с которой данные могут передаваться в память или из нее. Для памяти с произвольным доступом она равна $1/T_ц$. Для других видов памяти скорость передачи определяется соотношением:

$$T_N = T_A + \frac{N}{R},$$

где T_N — среднее время считывания или записи N битов; T_A — среднее время доступа; R — скорость пересылки в битах в секунду.

Говоря о *физическом типе* запоминающего устройства, необходимо упомянуть три наиболее распространенных технологии ЗУ — это полупроводниковая память, память с магнитным носителем информации, используемая в магнитных дисках и лентах, и память с оптическим носителем — оптические диски.

В зависимости от примененной технологии следует учитывать и ряд *физических особенностей* ЗУ, например энергозависимость. В энергозависимой памяти информация может быть искажена или потеряна при отключении источника питания. В энергонезависимых ЗУ записанная информация сохраняется и при отключении питающего напряжения. Магнитная и оптическая память — энергонезависимы. Полупроводниковая память может быть как энергозависимой, так и нет, в зависимости от ее типа. Помимо энергозависимости нужно учитывать, приводит ли считывание информации к ее разрушению.

Стоимость ЗУ принято оценивать отношением общей стоимости ЗУ к его емкости в битах, то есть стоимостью хранения одного бита информации.

Иерархия запоминающих устройств

Память часто называют «узким местом» фон-неймановских ВМ из-за ее серьезного отставания по быстродействию от процессоров, причем разрыв этот неуклонно увеличивается. Так, если производительность процессоров ежегодно возрастает вдвое примерно каждые 1,5 года, то для микросхем памяти прирост быстродействия не превышает 9% в год (удвоение за 10 лет), что выражается в увеличении разрыва в быстродействии между процессором и памятью приблизительно на 50% в год.

Если проанализировать используемые в настоящее время типы ЗУ, выявляется следующая закономерность:

- чем меньше время доступа, тем выше стоимость хранения бита;
- чем больше емкость, тем ниже стоимость хранения бита, но больше время доступа.

При создании системы памяти постоянно приходится решать задачу обеспечения требуемой емкости и высокого быстродействия за приемлемую цену. Наиболее распространенным подходом здесь является построение системы памяти ВМ по иерархическому принципу. *Иерархическая память* состоит из ЗУ различных типов (рис. 5.1), которые, в зависимости от характеристик, относят к определенному уровню иерархии. Более высокий уровень меньше по емкости, быстрее и имеет большую стоимость в пересчете на бит, чем более низкий уровень. Уровни иерархии взаимосвязаны: все данные на одном уровне могут быть также найдены на более низком уровне, и все данные на этом более низком уровне могут быть найдены на следующем нижележащем уровне и т. д.

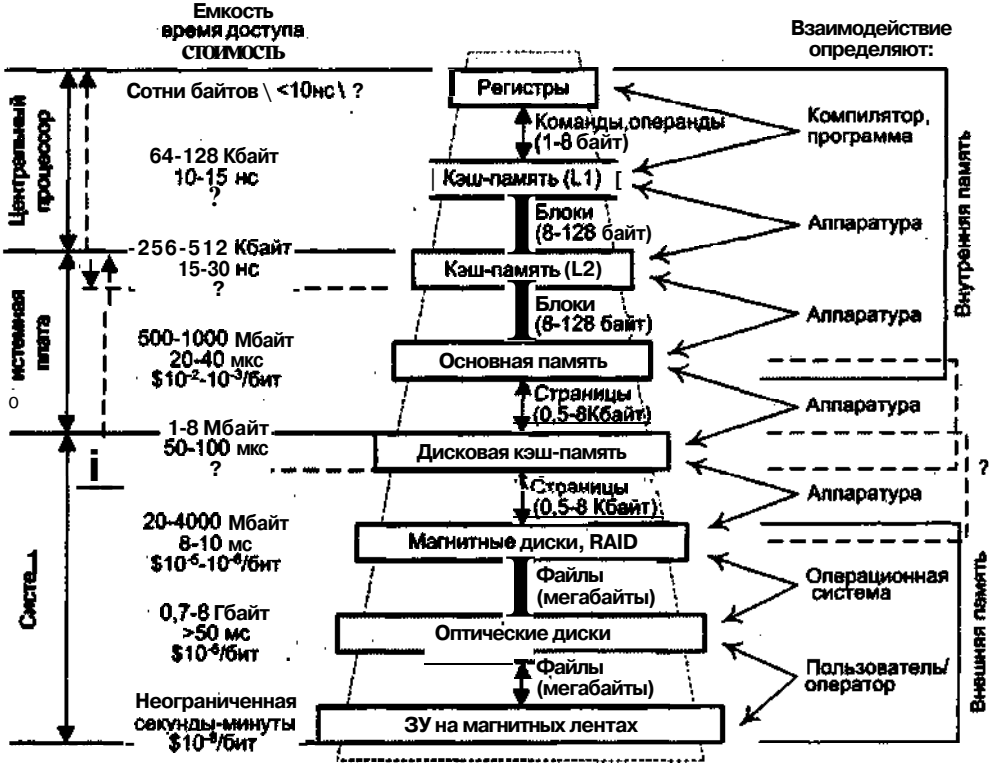


Рис. 5.1 . Иерархия запоминающих устройств

Четыре верхних уровня иерархии образуют *внутреннюю память* ВМ, а все нижние уровни - это *внешняя* или *вторичная* память. По мере движения вниз по иерархической структуре:

1. Уменьшается соотношение «стоимость/бит».
2. Возрастает емкость.
3. Растет время доступа.
4. Уменьшается частота обращения к памяти со стороны центрального процессора

Если память организована в соответствии с пунктами 1-3, а характер размещения в ней данных и команд удовлетворяет пункту 4, иерархическая организация ведет к уменьшению общей стоимости при заданном уровне производительности.

Справедливость этого утверждения вытекает из принципа *локальности по обращению* [87]. Если рассмотреть процесс выполнения большинства программ, то можно заметить, что с очень высокой вероятностью адрес очередной команды программы либо следует непосредственно за адресом, по которому была считана текущая команда, либо расположен вблизи него. Такое расположение адресов называется *пространственной локальностью программы*. Обрабатываемые данные, как правило, структурированы, и такие структуры обычно хранятся в последовательных ячейках памяти. Данная особенность программ называется *пространственной локальностью данных*. Кроме того, программы содержат множество небольших циклов и подпрограмм. Это означает, что небольшие наборы команд могут многократно повторяться в течение некоторого интервала времени, то есть имеет место *временная локальность*. Все три вида локальности объединяет понятие *локальность по обращению*. Принцип локальности часто облекают в численную форму и представляют в виде так называемого правила «90/10»: 90% времени работы программы связано с доступом к 10% адресного пространства этой программы.

Из свойства локальности вытекает, что программу разумно представить в виде последовательно обрабатываемых фрагментов - компактных групп команд и данных. Помещая такие фрагменты в более быструю память, можно существенно снизить общие задержки на обращение, поскольку команды и данные, будучи один раз переданы из медленного ЗУ в быстрое, затем могут использоваться многократно, и среднее время доступа к ним в этом случае определяется уже более быстрым ЗУ. Это позволяет хранить большие программы и массивы данных на медленных, емких, но дешевых ЗУ, а в процессе обработки активно использовать сравнительно небольшую быструю память, увеличение емкости которой сопряжено с высокими затратами.

На каждом уровне иерархии информация разбивается на *блоки,* выступающие в качестве наименьшей информационной единицы, пересылаемой между двумя соседними уровнями иерархии. Размер блоков может быть фиксированным либо переменным. При фиксированном размере блока емкость памяти обычно кратна его размеру. Размер блоков на каждом уровне иерархии чаще всего различен и увеличивается от верхних уровней к нижним.

При доступе к командам и данным, например, для их считывания, сначала производится поиск в памяти верхнего уровня. Факт обнаружения нужной информации называют *попаданием* (hit), в противном случае говорят о *промахе* (miss). При промахе производится поиск в. ЗУ следующего более низкого уровня, где также возможны попадание или промах. После обнаружении необходимой информации выполняется последовательная пересылка блока, содержащего искомую инфор-

мацию, с нижних уровней на верхние. Следует отметить, что независимо от числа уровней иерархии пересылка информации может осуществляться только между двумя соседними уровнями.

При оценке эффективности подобной организации памяти обычно используют следующие характеристики:

- *коэффициент попаданий* (hit rate) - отношение числа обращений к памяти, при которых произошло попадание, к общему числу обращений к ЗУ данного уровня иерархии;
- *коэффициент промахов* (miss rate) - отношение числа обращений к памяти, при которых имел место промах; к общему числу обращений к ЗУ данного уровня иерархии;
- *время обращения при попадании* (hit time) - время, необходимое для поиска нужной информации в памяти верхнего уровня (включая выяснение, является ли обращение попаданием), плюс время на фактическое считывание данных;
- *потери на промах* (miss penalty) — время, требуемое для замены блока в памяти более высокого уровня на блок с нужными данными, расположенный в ЗУ следующего (более низкого) уровня. Потери на промах включают в себя: *время доступа* (access time) - время обращения к первому слову блока при промахе и *время пересылки* (transfer time) — дополнительное время для пересылки оставшихся слов блока. Время доступа обусловлено задержкой памяти более низкого уровня, в то время как время пересылки связано с полосой пропускания канала между ЗУ двух смежных уровней.

Описание некоторого уровня иерархии ЗУ предполагает конкретизацию четырех моментов:

- размещения блока — допустимого места расположения блока на примыкающем сверху уровне иерархии;
- идентификации блока - способа нахождения блока на примыкающем сверху уровне;
- замещения блока - выбора блока, заменяемого при промахе с целью освобождения места для нового блока;
- согласования копий (стратегии записи) - обеспечения согласованности копий одних и тех же блоков, расположенных на разных уровнях, при записи новой информации в копию, находящуюся на более высоком уровне.

Самый быстрый, но и минимальный по емкости тип памяти - это внутренние регистры ЦП, которые иногда *объединяют понятием сверхоперативное запоминающее устройство* - СОЗУ. Как правило, количество регистров невелико, хотя в архитектурах с сокращенным набором команд их число может доходить до нескольких сотен. Основная память (ОП), значительно большей емкости, располагается несколькими уровнями ниже. Между регистрами ЦП и основной памятью часто размещают кэш-память, которая по емкости ощутимо проигрывает ОП, но существенно превосходит последнюю по быстродействию, уступая в то же время СОЗУ. В большинстве современных ВМ имеется несколько уровней кэш-памяти, которые обозначают буквой L и номером уровня кэш-памяти. На рис. 5.1 показаны

два таких уровня. В последних разработках все чаще появляется также третий уровень кэш-памяти (L3), причем разработчики ВМ говорят о целесообразности введения и четвертого уровня — L4. Каждый последующий уровень кэш-памяти имеет большую емкость, но одновременно и меньшее быстродействие по сравнению с предыдущим. Как бы то ни было, по «скорости» любой уровень кэш-памяти превосходит основную память. Все виды внутренней памяти реализуются на основе полупроводниковых технологий и в основном,являются энергозависимыми.

Долговременное хранение больших объемов информации (программ и данных) обеспечивается внешними ЗУ, среди которых наиболее распространены запоминающие устройства на базе магнитных и оптических дисков, а также магнитоленточные ЗУ.

Наконец, еще один уровень иерархии может быть добавлен между основной памятью и дисками. Этот уровень носит название дисковой кэш-памяти и реализуется в виде самостоятельного ЗУ, включаемого в состав магнитного диска. Дисковая кэш-память существенно улучшает производительность при обмене информацией между дисками и основной памятью.

Иерархия может быть дополнена и другими видами памяти. Так, некоторые модели ВМ фирмы IBM включают в себя так называемую расширенную память (expanded storage), выполненную на основе полупроводниковой технологии, но имеющую меньшее быстродействие и стоимость по сравнению с ОП. Строго говоря, этот вид памяти не входит в иерархию, а представляет собой ответвление от нее, поскольку данные могут передаваться только между расширенной и основной памятью, но не допускается обмен между расширенной и внешней памятью.

Основная память

Основная память (ОП) представляет собой единственный вид памяти, к которой ЦП может обращаться непосредственно (исключение составляют лишь регистры центрального процессора). Информация, хранящаяся на внешних ЗУ, становится доступной процессору только после того, как будет переписана в основную память.

Основную память образуют запоминающие устройства с произвольным доступом. Такие ЗУ образованы как массив ячеек, а «произвольный доступ» означает, что обращение к любой ячейке занимает одно и то же время и может производиться в произвольной последовательности. Каждая ячейка содержит фиксированное число запоминающих элементов и имеет уникальный адрес, позволяющий различать ячейки при обращении к ним для выполнения операций записи и считывания.

Следствием огромных успехов в области полупроводниковых технологий стало изменение элементной базы основной памяти. На смену ЗУ на базе ферромагнитных колец пришли полупроводниковые микросхемы, использование которых в наши дни стало повсеместным[1].

[1] В полупроводниковых ЗУ применяются запоминающие элементы на основе: биполярных транзисторов; приборов со структурой «металл-окисел-полупроводник» (МОП-транзисторов); приборов со структурой «металл-нитрид-окисел-полупроводник» (МНОП); приборов с зарядовой связью (ПЗС); МОП-транзисторов с изолированным затвором и др.

Основная память может включать в себя два типа устройств: *оперативные запоминающие устройства* (ОЗУ) и *постоянные запоминающие устройства* (ПЗУ).

Преимущественную долю основной памяти образует ОЗУ, называемое оперативным, потому что оно допускает как запись, так и считывание информации, причем обе операции выполняются однотипно, практически с одной и той же скоростью, и производятся с помощью электрических сигналов. В англоязычной литературе ОЗУ соответствует аббревиатура RAM — *Random Access Memory,* то есть «память с произвольным доступом», что не совсем корректно, поскольку памятью с произвольным доступом являются также ПЗУ и регистры процессора. Для большинства типов полупроводниковых ОЗУ характерна энергозависимость — даже при кратковременном прерывании питания хранимая информация теряется. Микросхема ОЗУ должна быть постоянно подключена к источнику питания и поэтому может использоваться только как временная память.

Вторую группу полупроводниковых ЗУ основной памяти образуют энергонезависимые микросхемы ПЗУ (ROM - *Read-Only Memory).* ПЗУ обеспечивает считывание информации, но не допускает ее изменения (в ряде случаев информация в ПЗУ может быть изменена, но этот процесс сильно отличается от считывания и требует значительно большего времени).

Блочная организация основной памяти

Емкость **основной памяти** современных ВМ слишком велика, чтобы ее можно было реализовать на базе единственной интегральной микросхемы (ИМС). Необходимость объединения нескольких ИМС ЗУ возникает также, когда разрядность ячеек в микросхеме ЗУ меньше разрядности слов ВМ.

Увеличение разрядности ЗУ реализуется за счет объединения адресных входов объединяемых ИМС ЗУ. Информационные входы и выходы микросхем являются входами и выходами модуля ЗУ увеличенной разрядности (рис. 5.2). Полученную совокупность микросхем называют *модулем памяти* Модулем можно считать и единственную микросхему, если она уже имеет нужную разрядность. Один или несколько модулей образуют *банк памяти.*

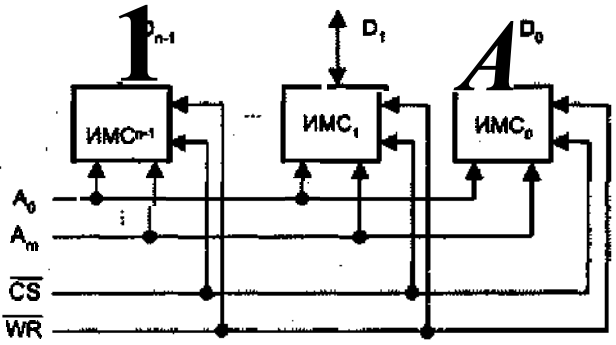


Рис. 5.2. Увеличение разрядности памяти

Для получения требуемой емкости ЗУ нужно определенным образом объединить несколько банков памяти меньшей емкости. В общем случае основная па-

мять ВМ практически всегда имеет блочную структуру, то есть содержит несколько банков.

При использовании блочной памяти, состоящей из *B* банков, адрес ячейки *A* преобразуется в пару *(Ъ, w)*, где *Ъ* - номер банка, *w* - адрес ячейки внутри банка. Известны три схемы распределения разрядов адреса *A* между *b* и *w*:

- блочная (номер банка *Ъ* определяет старшие разряды адреса);
- циклическая *(Ъ=A* mod *B*: *w = A div B);*
- блочно-циклическая (комбинациядвух предыдущих схем).

Рассмотрение основных структур блочной ОП будем проводить на примере памяти емкостью 512 слов (2^9), построенной из четырех банков по 128 слов в каждом. Типовая структура памяти, организованная в соответствии с блочной структурой, показана на рис. 5.3.

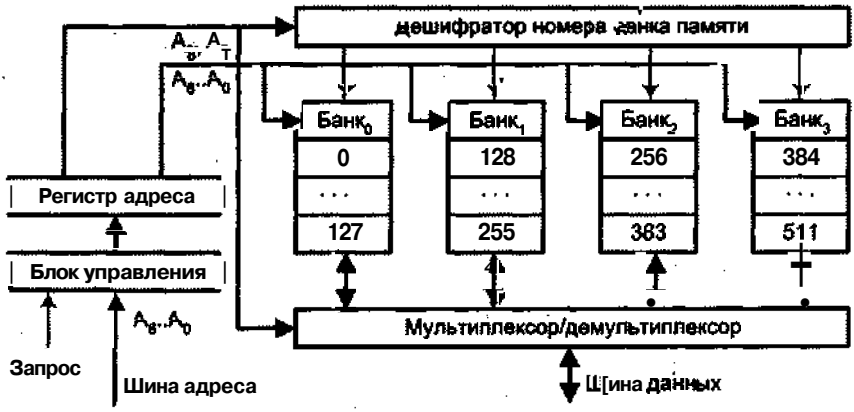


Рис. 5.3. Структура основной памяти на основе блочной схемы

Адресное пространство памяти разбито на группы последовательных адресов, и каждая такая группа обеспечивается отдельным банком памяти. Для обращения к ОП используется 9-разрядный адрес, семь младших разрядов которого *(A_6-A)* поступают параллельно на все банки памяти и выбирают в каждом из них одну ячейку. Два старших разряда адреса *(A_8, A_7)* содержат номер банка. Выбор банка обеспечивается либо с помощью дешифратора номера банка памяти, либо путем мультиплексирования информации (на рис. 5.3 показаны оба варианта). В функциональном отношении такая ОП может рассматриваться как единое ЗУ, емкость которого равна суммарной емкости составляющих, а быстродействие - быстродействию отдельного банка.

Расслоение памяти

Помимо податливости к наращиванию емкости, блочное построение памяти обладает еще одним достоинством — позволяет сократить время доступа к информации. Это возможно благодаря потенциальному параллелизму, присущему блочной организации. Большей скорости доступа можно достичь за счет одновременного доступа ко многим банкам памяти. Одна из используемых для этого методик на-

зывается расслоением памяти. В ее основе лежит так называемое *чередование адресов* (address interleaving), заключающееся в изменении системы распределения адресов между банками памяти.

Прием чередования адресов базируется на ранее рассмотренном свойстве локальности по обращению, согласно которому последовательный доступ в память обычно производится к ячейкам, имеющим смежные адреса. Иными словами, если в данный момент выполняется обращение к ячейке с адресом 5, то следующее обращение, вероятнее всего, будет к ячейке с адресом 6, затем 7 и т. д. Чередование адресов обеспечивается за счет циклического разбиения адреса. В нашем примере (рис. 5.4) для выбора банка используются два младших разряда адреса (A_1, A_0), а для выбора ячейки в банке - 7 старших разрядов $(A_8–A_2)$.

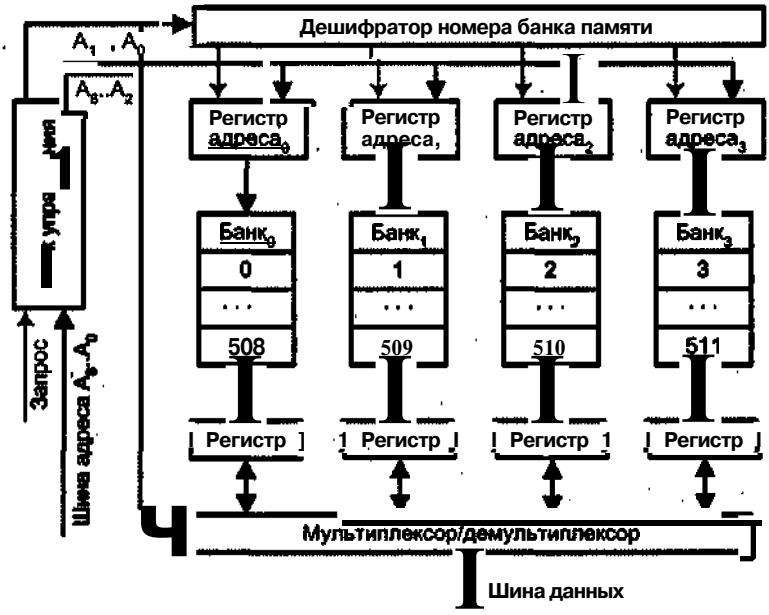


Рис. 5.4. Блочная память с чередованием адресов по циклической схеме

Поскольку в каждом такте на шине адреса может присутствовать адрес только одной ячейки, параллельное обращение к нескольким банкам невозможно, однако оно может быть организовано со сдвигом на один такт. Адрес ячейки запоминается в индивидуальном регистре адреса, и дальнейшие операции по доступу к ячейке в каждом банке протекают независимо. При большом количестве банков среднее время доступа к ОП сокращается почти в *B* раз *(B* — количество банков), но при условии, что ячейки, к которым производится последовательное обращение, относятся к разным банкам. Если же запросы к одному и тому же банку следуют друг за другом, каждый следующий запрос должен ожидать завершения обслуживания предыдущего. Такая ситуация называется *конфликтом по доступу*. При частом возникновении конфликтов по доступу метод становится неэффективным.

В блочно-циклической схеме расслоения памяти каждый банк состоит из нескольких модулей, адресуемых по круговой схеме. Адреса между банками распре-

делены по блочной схеме. Таким образом, адрес ячейки разбивается на три части. Старшие биты определяют номер банка, следующая группа разрядов адреса указывает на ячейку в модуле, а младшие биты адреса выбирают модуль в банке. Схему иллюстрирует рис. 5.5.

Рис. 5.5. Блочно-циклическая схема расслоения памяти

Традиционные способы расслоения памяти хорошо работают в рамках одной задачи, для которой характерно свойство локальности. В многопроцессорных системах с общей памятью, где запросы на доступ к памяти достаточно независимы, не исключен иной подход, который можно рассматривать как развитие идеи расслоения памяти. Для этого в систему включают несколько контроллеров памяти, что позволяет отдельным банкам работать совершенно автономно. Эффективность данного приема зависит от частоты независимых обращений к разным банкам. Лучшего результата можно ожидать при большом числе банков, что уменьшает вероятность последовательных обращений к одному и тому же банку памяти. Так, в суперкомпьютере NEC SX/3 основная память состоит из 128 банков.

Организация микросхем памяти

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

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

При матричной организации ИМС памяти (рис. 5.6) реализуется координатный принцип адресации ячеек. Адрес ячейки, поступающий по шине адреса ВМ, пропускается через логику выбора, где он разделяется на две составляющие: адрес строки и адрес столбца. Адреса строки и столбца запоминаются соответственно в регистре адреса строки и регистре адреса столбца микросхемы. Регистры соединены каждый со своим дешифратором. Выходы дешифраторов образуют систему горизонтальных и вертикальных линий, к которым подсоединены запоминающие элементы матрицы, при этом каждый ЗЭ расположен на пересечении одной горизонтальной и одной вертикальной линии.

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

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

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

Для уменьшения числа контактов ИМС адреса строки и столбца в большинстве микросхем подаются в микросхему через одни и те же контакты последовательно во времени (мультиплексируются) и запоминаются соответственно в регистре адреса строки и регистре адреса столбца микросхемы. Мультиплексирование обычно реализуется внешней по отношению к ИМС схемой.

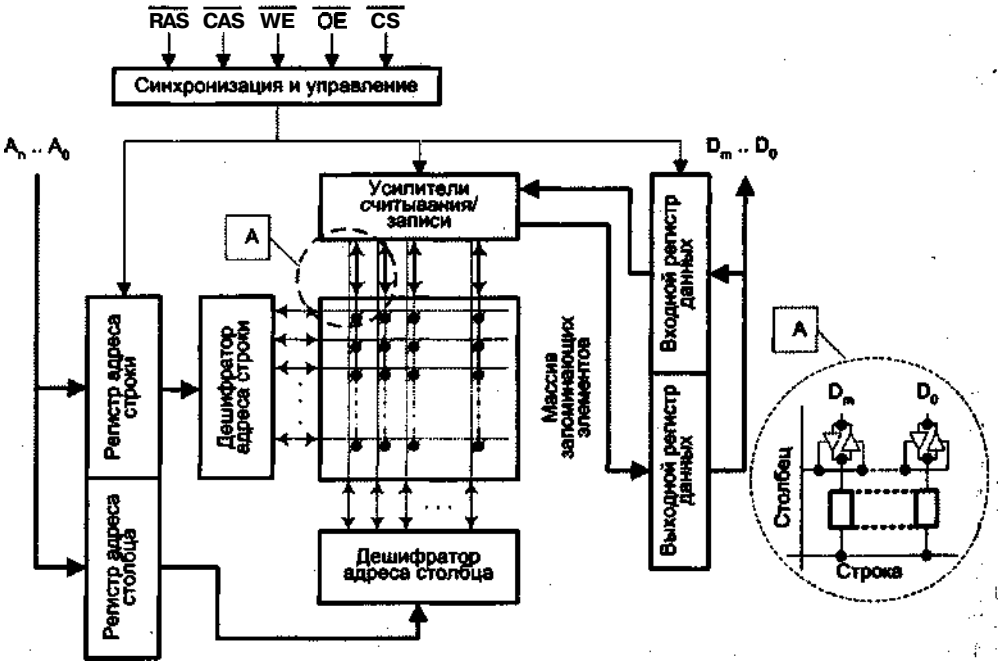


Рис. 5.6. Структура микросхемы памяти

Для синхронизации процессов фиксации и обработки адресной информации внутри ИМС адрес строки (RA) сопровождается сигналом RAS (Row Address Strobe - строб строки), а адрес столбца (CA) - сигналом CAS (Column Address Strobe - строб столбца). Вторую букву в аббревиатурах RAS и CAS иногда расшифровывают как Access - «доступ», то есть имеется строб доступа к строке и строб доступа к столбцу. Чтобы стробирование было надежным, эти сигналы подаются с задержкой, достаточной для завершения переходных процессов на шине адреса и в адресных цепях микросхемы.

Сигнал выбора микросхемы CS (Crystal Select) разрешает работу ИМС и используется для выбора определенной микросхемы в системах, состоящих из нескольких ИМС. Вход WE (Write Enable - разрешение записи) определяет вид выполняемой операции (считывание или запись).

Записываемая информация, поступающая по шине данных, первоначально заносится во входной регистр данных, а затем - в выбранную ячейку. При выполнении операции чтения информация из ячейки до ее выдачи на шину данных буферизируется в выходном регистре данных. Обычно роль входного и выходного выполняет один и тот же регистр. Усилители считывания/записи (УСЗ) служат для электрического согласования сигналов на линиях данных и внутренних сигналов ИМС. Обычно число УСЗ равно числу запоминающих элементов в строке матрицы, и все они при обращении к памяти подключаются к выбранной горизонтальной линии. Каждая группа УСЗ, образующая ячейку, подключена к одному из столбцов матрицы, то есть выбор нужной ячейки в строке обеспечивается активизацией одной из вертикальных линий. На все время пока ИМС памяти не использует шину данных, информационные выходы микросхемы переводятся в третье (высокоимпедансное) состояние. Управление переключением в третье состояние обеспечивается сигналом OE (Output Enable - разрешение выдачи выходных сигналов). Этот сигнал активизируется при выполнении операции чтения.

Для большинства перечисленных выше управляющих сигналов активным обычно считается их низкий уровень, что и показано на рис. 5.6.

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

1. Указание типа операции (чтение или запись) и установка адреса строки.
2. Формирование сигнала RAS.
3. Установка адреса столбца.
4. Формирование сигнала CAS.
5. Возврат сигналов RAS и CAS в неактивное состояние.

Данный перечень учитывает далеко не все необходимые действия, например регенерацию содержимого памяти в динамических ОЗУ.

Типовую процедуру доступа к памяти рассмотрим на примере чтения из ИМС с мультиплексированием адресов строк и столбцов. Сначала на входе WE устанавливается уровень, соответствующий операции чтения, а на адресные контакты ИМС подается адрес строки, сопровождаемый сигналом RAS. По заднему фронту этого сигнала адрес запоминается в регистре адреса строки микросхемы, после чего дешифрируется. После стабилизации процессов, вызванных сигналом RAS, выбранная строка подключается к УСЗ. Далее на вход ИМС подается адрес столбца, который по заднему фронту сигнала CAS заносится в регистр адреса столбца. Одновременно подготавливается выходной регистр данных, куда после стабилизации сигнала CAS загружается информация с выбранных УСЗ.

Разработчики микросхем памяти тратят значительные усилия на повышение быстродействия ИМС, которое принято характеризовать четырьмя параметрами (численные значения приводятся для типовой микросхемы динамической памяти емкостью 4 Мбит):

- t_{RAS} — минимальное время от перепада сигнала RAS с высокого уровня к низкому до момента появления и стабилизации считанных данных на выходе ИМС. Среди приводившихся в начале главы характеристик быстродействия это соответствует *времени доступа* $T_Д$ (t_{RAS}=60 нс);
- t_{RC} — минимальное время от начала доступа к одной строке микросхемы памяти до Начала доступа к следующей строке. Этот параметр также упоминался в начале главы как *длительность цикла памяти* $T_Ц$ (t_{RC}=110 нс при t_{RAS}=60 нс);
- t_{CAS} — минимальное время от перепада сигнала CAS с высокого уровня к низкому до момента появления и стабилизации считанных данных на выходе ИМС (t_{CAS}= 15 нс при t_{RAS} 60 нс);
- T_{PC} — минимальное время от начала доступа к одному столбцу микросхемы памяти до начала доступа к следующему столбцу (t_{PC}=35 нс при t_{RAS}= 60 нс).

Возможности «ускорения ядра» микросхемы ЗУ весьма ограничены и связаны в основном с миниатюризацией запоминающих элементов. Наибольшие успехи достигнуты в интерфейсной части ИМС, касаются они, главным образом, операции чтения, то есть способов доставки содержимого ячейки на шину данных. Наибольшее распространение получили следующие шесть фундаментальных подходов:

- последовательный;
- конвейерный;
- регистровый;
- страничный;
- пакетный;
- удвоенной скорости.

Последовательный режим

При использовании *последовательного режима* (Flow through Mode) адрес и управляющие сигналы подаются на микросхему до поступления синхроимпульса

В момент прихода синхроимпульса вся входная информация запоминается во внутренних регистрах — по его переднему фронту, и начинается цикл чтения. Через некоторое время, но в пределах того же цикла данные появляются на внешней шине, причем момент этот определяется только моментом прихода синхронизирующего импульса и скоростью внутренних цепей микросхемы.

Конвейерный режим

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

При чтении из памяти время, требуемое для извлечения данных из ячейки, можно условно разбить на два интервала. Первый из них — непосредственно доступ к массиву запоминающих элементов и извлечение данных из ячейки. Второй — передача данных на выход (при этом происходит детектирование состояния ячейки, усиление сигнала и другие операции, необходимые для считывания информации).

В отличие от последовательного режима, где следующий цикл чтения начинается только по окончании предыдущего, в конвейерном режиме процесс разбивается на два этапа. Пока данные из предыдущего цикла чтения передаются на внешнюю шину, происходит запрос на следующую операцию чтения. Таким образом, два цикла чтения перекрываются во времени. Из-за усложнения схемы передачи данных на внешнюю шину время считывания увеличивается на один такт, и данные поступают на выход только в следующем такте, но такое запаздывание наблюдается лишь при первом чтении в последовательности операций считывания из памяти. Все последующие данные поступают на выход друг за другом, хотя и с запаздыванием на один такт относительно запроса на чтение. Так как циклы чтения перекрываются, микросхемы с конвейерным режимом могут использоваться при частотах шины, вдвое превышающих допустимую для ИМС с последовательным режимом чтения.

Регистровый режим

Регистровый режим (Register to Latch) используется относительно редко и отличается наличием регистра на выходе микросхемы. Адрес и управляющие сигналы выдаются на шину до поступления синхронизирующего импульса. С приходом положительного фронта синхроимпульса адрес записывается во внутренний регистр микросхемы, и начинается цикл чтения. Считанные данные заносятся в промежуточный выходной регистр и хранятся там до появления отрицательного фронта (спада) синхроимпульса, а с его поступлением передаются на шину. Метод однозначно определяет момент появления данных на выходе ИМС, причем изменяя ширину импульса синхронизации можно менять время появления данных на шине. Данное свойство часто оказывается весьма полезным при проектировании специализированных ВМ. По быстродействию микросхемы с регистровым режимом идентичны ИМС с последовательным режимом.

Страничный режим

В основе идеи лежит тот факт, что при доступе к ячейкам со смежными адресами (согласно принципу локальности такая ситуация наиболее вероятна), причем

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

Режим быстрого страничного доступа

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

Пакетный режим

Пакетный режим (Burst Mode) — режим, при котором на запрос по конкретному адресу память возвращает пакет данных, хранящихся не только по этому адресу, но и по нескольким последующим адресам.

Разрядность ячейки памяти современных ВМ обычно равна одному байту, в то время как ширина шины данных, как правило, составляет четыре байта. Следовательно , одно обращение к памяти требует последовательного доступа к четырем смеж-

ным ячейкам - пакету[1]. С учетом этого обстоятельства в ИМС памяти часто используется модификация страничного режима, носящая название *группового* или *пакетного режима*. При его реализации адрес столбца заносится в ИМС только для первой ячейки пакета, а переход к очередному столбцу производится уже внутри микросхемы. Это позволяет для каждого пакета исключить три из четырех операций занесения в ИМС адреса столбца и тем самым еще более сократить среднее время доступа.

Режим удвоенной скорости

Важным этапом в дальнейшем развитии технологии микросхем памяти стал режим DDR (Double Data Rate) - удвоенная скорость передачи данных. Сущность метода заключается в передаче данных по обоим фронтам импульса синхронизации, то есть дважды за период. Таким образом, пропускная способность увеличивается в те же два раза.

Помимо упомянутых используются и другие приемы повышения быстродействия ИМС памяти, такие как включение в состав микросхемы вспомогательной кэш-памяти и независимые тракты данных, позволяющие одновременно производить обмен с шиной данных и обращение к матрице ЗЭ и т. д.

Синхронные и асинхронные запоминающие устройства

В качестве первого критерия, по которому можно классифицировать запоминающие устройства основной памяти, рассмотрим способ синхронизации. С этих позиций известные типы ЗУ подразделяются на синхронные и асинхронные.

В микросхемах, где реализован *синхронный принцип,* процессы чтения и записи (если это ОЗУ) выполняются одновременно с тактовыми сигналами контроллера памяти.

Асинхронный принцип предполагает, что момент начала очередного действия определяется только моментом завершения предшествующей операции. Переносятся этот принцип на систему памяти, необходимо принимать во внимание, что контроллер памяти всегда работает синхронно. В *асинхронных ЗУ* цикл чтения начинается только при поступлении запроса от контроллера памяти, и если память не успевает выдать данные в текущем такте, контроллер может считать их только в следующем такте, поскольку очередной шаг контроллера начинается с приходом очередного тактового импульса. В последнее время асинхронная схема активно вытесняется синхронной.

Оперативные запоминающие устройства

Большинство из применяемых в настоящее время типов микросхем оперативной памяти не в состоянии сохранять данные без внешнего источника энергии, то есть являются энергозависимыми (volatile memory). Широкое распространение таких устройств связано с рядом их достоинств по сравнению с энергонезависимыми

[1] Строго говоря, количество ячеек, считываемое за один раз без дополнительного указания адреса и называемое длиной пакета (burst length), в большинстве случаев может программироваться. Помимо упомянутых четырех это могут быть 1, 2 или 8 ячеек подряд.

типами ОЗУ (non-volatile memory): большей емкостью, низким энергопотреблением, более высоким быстродействием и невысокой себестоимостью хранения единицы информации.

Энергозависимые ОЗУ можно подразделить на две основные подгруппы: динамическую память (DRAM - Dynamic Random, Access Memory) и статическую память (SRAM - Static Random Access Memory).

Статическая и динамическая оперативная память

В *статических ОЗУ* запоминающий элемент может хранить записанную информацию неограниченно долго (при наличии питающего напряжения). Запоминающий элемент *динамического ОЗУ* способен хранить информацию только в течение достаточно короткого промежутка времени, после которого информацию нужно восстанавливать заново, иначе она будет потеряна. Динамические ЗУ, как и статические, энергозависимы.

Роль запоминающего элемента в статическом ОЗУ исполняет триггер. Такой триггер представляет собой схему с двумя устойчивыми состояниями, обычно состоящую из четырех или шести транзисторов (рис. 5.7). Схема с четырьмя транзисторами обеспечивает большую емкость микросхемы, а следовательно, меньшую стоимость, однако у такой схемы большой ток утечки, когда информация просто хранится. Также триггер на четырех транзисторах более чувствителен к воздействию внешних источников излучения, которые могут стать причиной потери информации. Наличие двух дополнительных транзисторов позволяет в какой-то мере компенсировать упомянутые недостатки схемы на четырех транзисторах, но, главное - увеличить быстродействие памяти.

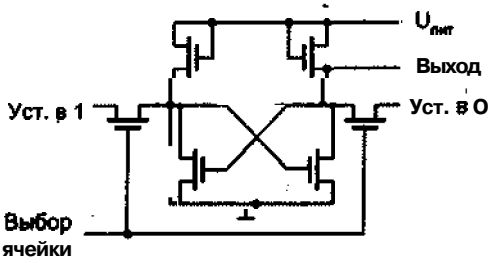


Рис. 5.7. Запоминающий элемент статического ОЗУ

Запоминающий элемент динамической памяти значительно проще. Он состоит из одного конденсатора и запирающего транзистора (рис. 5.8).

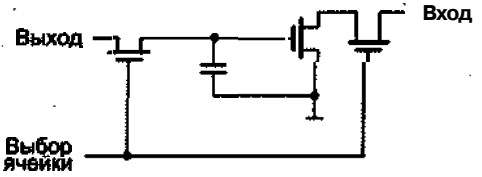


Рис. 5.8. Запоминающий элемент динамического ОЗУ

Наличие или отсутствие заряда в конденсаторе интерпретируется как 1 или О соответственно. Простота схемы позволяет достичь высокой плотности размещения ЗЭ и, в итоге, снизить стоимость. Главный недостаток подобной технологии связан с тем, что накапливаемый на конденсаторе заряд со временем теряется. Даже при хорошем диэлектрике с электрическим сопротивлением в несколько тераом (10^{12} Ом), используемом при изготовлении элементарных конденсаторов ЗЭ, заряд теряется достаточно быстро. Размеры у такого конденсатора микроскопические, а емкость имеет порядок 10^{-15} Ф. При такой емкости на одном конденсаторе накапливается всего около 40 000 электронов. Среднее время утечки заряда ЗЭ динамической памяти составляет сотни или даже десятки миллисекунд, поэтому заряд необходимо успеть восстановить в течение данного отрезка времени, иначе хранящаяся информация будет утеряна. Периодическое восстановление заряда ЗЭ называется *регенерацией* и осуществляется каждые2- 8 мс.

В различных типах ИМС динамической памяти нашли применение три основных метода регенерации:

- одним сигналом RAS (ROR - RAS Only Refresh);
- сигналом CAS, предваряющим сигнал RAS(CBR - CAS Before RAS);
- автоматическая регенерация (SR — Self Refresh).

Регенерация одним RAS использовалась еще в первых микросхемах DRAM. На шину адреса выдается адрес регенерируемой строки, сопровождаемый сигналом RAS. При этом выбирается строка ячеек и хранящиеся там данные поступают на внутренние цепи микросхемы, после чего записываются обратно. Так как сигнал CAS не появляется, цикл чтения/записи не начинается. В следующий раз на шину адреса подается адрес следующей строки и т. д., пока не восстановятся все ячейки, после чего цикл повторяется. К недостаткам метода можно отнести занятость шины адреса в момент регенерации, когда доступ к другим устройствам ВМ блокирован.

Особенность метода CBR в том, что если в обычном цикле чтения/записи сигнал RAS всегда предшествует сигналу CAS, то при появлении сигнала CAS первым начинается специальный цикл регенерации. В этом случае адрес строки не передается, а микросхема использует свой внутренний счетчик, содержимое которого увеличивается на единицу при каждом очередном CBR-цикле. Режим позволяет регенерировать память, не занимая шину адреса, то есть более эффективен.

Автоматическая регенерация памяти связана с энергосбережением, когда система переходит в режим «сна» и тактовый генератор перестает работать. При отсутствии внешних сигналов RAS и CAS обновление содержимого памяти методами ROR или CBR невозможно, и микросхема производит регенерацию самостоятельно, запуская собственный генератор, который тактирует внутренние цепи регенерации.

Область применения статической и динамической памяти определяется скоростью и стоимостью. Главным преимуществом SRAM является более высокое быстродействие (примерно на порядок выше, чем у DRAM). Быстрая синхронная SRAM может работать со временем доступа к информации, равным времени одного тактового импульса процессора. Однако из-за малой емкости микросхем и высо-

кой стоимости применение статической памяти, как правило, ограничено относительно небольшой по емкости кэш-памятью первого (L1), второго (L2) или третьего (L3) уровней. В то же время самые быстрые микросхемы динамической памяти на чтение первого байта пакета все еще требуют от пяти до десяти тактов процессора, что замедляет работу всей ВМ. Тем не менее благодаря высокой плотности упаковки ЗЭ и низкой стоимости именно DRAM используется при построении основной памяти ВМ.

Статические оперативные запоминающие устройства

Напомним, что роль запоминающего элемента в статическом ОЗУ исполняет триггер. Статические ОЗУ на настоящий момент - наиболее быстрый, правда, и наиболее дорогостоящий вид оперативной памяти. Известно достаточно много различных вариантов реализации SRAM, отличающихся по технологии, способам организации и сфере применения (рис. 5.9).

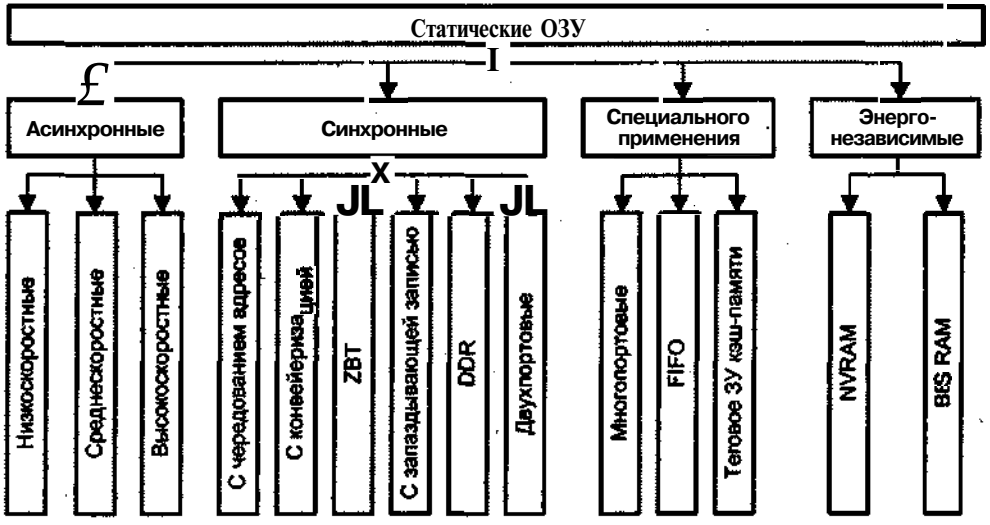


Рис. 5.9. Виды статических ОЗУ

Асинхронные статические ОЗУ. Асинхронные статические ОЗУ применялись в кэш-памяти второго уровня в течение многих лет, еще с момента появления микропроцессора i80386. Для таких ИМС время доступа составляло 15-20 нс (в лучшем случае — 12 нс), что не позволяло кэш-памяти второго уровня работать в темпе процессора.

Синхронные статические ОЗУ. В рамках данной группы статических ОЗУ выделяют ИМС типа SSRAM и более совершенные PB SRAM.

Значительно лучшие показатели по сравнению с асинхронными статическими ОЗУ достигнуты в синхронных SRAM (SSRAM). Как и в любой синхронной памяти, все события в SSRAM происходят с поступлением внешних тактовых импульсов. Отличительная особенность SSRAM - входные регистры, где фиксируется входная информация. Рассматриваемый вид памяти обеспечивает работу

в пакетном режиме с формулой 3-1-1-1, но лишь до определенных значений тактовой частоты шины. При более высоких частотах формула изменяется на 3-2-2-2.

Последние модификации микропроцессоров Pentium, начиная с Pentium II, взамен SSRAM оснащаются статической оперативной памятью с пакетным конвейерным доступом (PB SRAM - Pipelined Burst SRAM). В этой разновидности SRAM реализована внутренняя конвейеризация, за счет которой скорость обмена пакетами данных возрастает примерно вдвое. Память данного типа хорошо работает при повышенных частотах системной шины. Время доступа к PB SRAM составляет от 4,5 до 8 нс, при этом формула 3-1-1-1 сохраняется даже при частоте системной шины 133 МГц.

Особенности записи в статических ОЗУ. Важным моментом, характеризующим SRAM, является технология записи. Известны два варианта записи: *стандартная* и *запаздывающая*. В стандартном режиме адрес и данные выставляются на соответствующие шины в одном и том же такте. В режиме запаздывающей записи данные для нее передаются в следующем такте после выбора адреса нужной ячейки, что напоминает режим конвейерного чтения, когда данные появляются на шине в следующем такте. Оба рассматриваемых варианта позволяют производить запись данных с частотой системной шины. Различия сказываются только при переключении между операциями чтения и записи.

Более детально различия режимов записи в SRAM рассмотрим на примере выполнения конвейерного чтения из ячеек с адресами A0, A1 и A2 с последующей записью в ячейку с адресом A3.

В режиме стандартной записи перед выработкой первого импульса синхронизации (ИС) на шину адреса выдается адрес первой ячейки для чтения A0 С приходом первого ИС этот адрес записывается во внутренний регистр микросхемы, и начинается цикл чтения. Перед началом второго ИС на шину адреса выставляется адрес следующей ячейки A1, и начинается второй цикл чтения. В это время данные из ячейки A0 поступают на шину данных. На третьем этапе выставляется адрес A2, а данные из ячейки A1 приходят на шину. В четвертом тактовом периоде предполагается запись, перед началом которой информационные выходы ИМС должны быть переведены в третье (высокоимпедансное) состояние. В результате данные из ячейки A1, появившиеся на шине только в конце третьего тактового периода, будут находиться там недостаточно долго, чтобы их можно было использовать. Таким образом, в третьем тактовом периоде данные не считываются и не записываются, и этот период называют *холостым циклом*. С началом четвертого такта данные, выставленные на шине данных, записываются в ячейку с адресом A3. Адрес следующей ячейки для чтения можно выставить только в пятом тактовом периоде, а соответствующие данные будут получены в шестом, то есть происходит еще один холостой цикл. В итоге за четыре такта произведены считывание из ячейки A0 и запись в ячейку A3. Как видно из описания, режим стандартной записи предусматривает потерю нескольких тактов шины при переключении между циклами чтения и записи. Если такая память используется в качестве кэш-памяти, то это не слишком влияет на производительность ВМ, так как запись в кэш-память происходит гораздо реже, чем чтение, и переключения «чтение/запись» и «запись/чтение» возникают относительно редко.

В режиме запаздывающей записи данные, которые должны быть занесены в ячейку, выставляются на шину лишь в следующем тактовом периоде. При этом данные, которые считываются из ячейки A1 в третьем такте, находятся в активном состоянии на протяжении всего тактового периода и могут быть беспрепятственно считаны в то время, когда выставляется адрес A3. Сами данные для записи передаются в четвертом такте, где в режиме стандартной записи имеет место холостой цикл. Как следствие, здесь за те же четыре такта считано содержимое двух ячеек (A0 и A1) и записаны данные по адресу A3.

Как видно из вышеизложенного, в обоих случаях адрес A2 игнорируется. Реально никакой потери адресов и данных не происходит. Контроллер памяти непосредственно перед переключением из режима чтения в режим записи просто не передает адрес на шину, так как «знает», какой тип памяти используется и сколько тактов ожидания нужно ввести перед переходом «чтение/запись» и обратно.

Компания IDT (Integrated Device Technology) в развитие идеи записи с запаздыванием предложила новую технологию, получившую название ZBT SRAM (Zero Bus Turnaround) — нулевое время переключения шины. Идея ее состоит в том, чтобы запись с запаздыванием производить с таким же интервалом, какой требуется для чтения. Так, если SRAM с конвейерным чтением требует три тактовых периода для чтения данных из ячейки, то данные для записи нужно передавать с таким же промедлением относительно адреса. В результате перекрывающиеся циклы чтения и записи идут один за другим, позволяя выполнять операции чтения/записи в каждом такте без каких-либо задержек[1].

Динамические оперативные запоминающие устройства

Динамической памяти в вычислительной машине значительно больше, чем статической, поскольку именно DRAM используется в качестве основной памяти ВМ. Как и SRAM, динамическая память состоит из ядра (массива ЗЭ) и интерфейсной логики (буферных регистров, усилителей чтения данных, схемы регенерации и др.). Хотя количество видов DRAM уже превысило два десятка, ядро у них организовано практически одинаково. Главные различия связаны с интерфейсной логикой, причем различия эти обусловлены также и областью применения микросхем - помимо основной памяти ВМ, ИМС динамической памяти входят, например, в состав видеоадаптеров. Классификация микросхем динамической памяти показана на рис. 5.10.

Чтобы оценить различия между видами DRAM, предварительно остановимся на алгоритме работы с динамической памятью. Для этого воспользуемся рис. 5.6.

В отличие от SRAM адрес ячейки DRAM передается в микросхему за два шага - Вначале адрес столбца, а затем строки, что позволяет сократить количество выводов шины адреса примерно вдвое, уменьшить размеры корпуса и разместить на материнской плате большее количество микросхем. Это, разумеется, приводит к снижению быстродействия, так как для передачи адреса нужно вдвое больше времени. Для указания, какая именно часть адреса передается в определенный момент, служат два вспомогательных сигнала RAS и CAS. При обращении к ячейке памяти на шину адреса выставляется адрес строки. После стабилизации процессов на

[1] Сходную с ZBT SRAM технологию предложила также фирма Cypress Semiconductor. Эта технология получила название NoBL SRAM (No Bus Latency - дословно «нет задержек шины»).

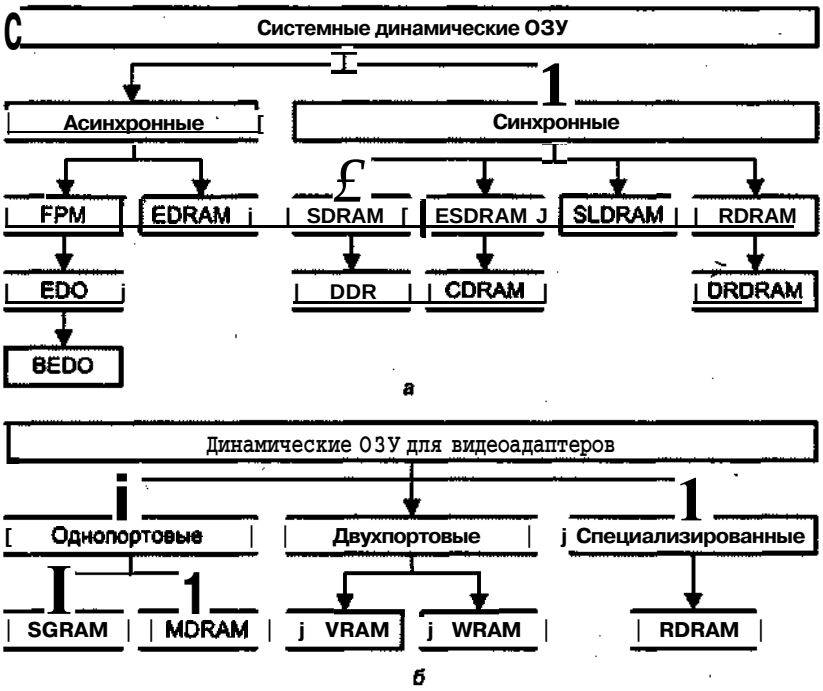


Рис. 5.10. Классификация динамических ОЗУ: а - микросхемы для основной памяти; *б* — микросхемы для.видеоадаптеров

шине подается сигнал RAS и адрес записывается во внутренний регистр микросхемы памяти. Затем на шину адреса выставляется адрес столбца и выдается сигнал CAS. В зависимости от состояния линии WE производится чтение данных из ячейки или их запись в ячейку (перед записью данные должны быть помещены на шину данных). Интервал между установкой адреса и выдачей сигнала RAS (или CAS) оговаривается техническими характеристиками микросхемы, но обычно адрес выставляется в одном такте системной шины, а управляющий сигнал - в следующем. Таким образом, для чтения или записи одной ячейки динамического ОЗУ требуется пять тактов, в которых происходит соответственно: выдача адреса строки, выдача сигнала RAS, выдача адреса столбца, выдача сигнала CAS, выполнение операции чтения/записи (в статической памяти процедура занимает лишь от двух до трех тактов).

Следует также помнить о необходимости регенерации данных. Но наряду с естественным разрядом конденсатора ЗЭ со временем к потере заряда приводит также считывание данных из DRAM, поэтому после каждой операции чтения данные должны быть восстановлены. Это достигается за счет повторной записи тех же данных сразу после чтения. При считывании информации из одной ячейки фактически выдаются данные сразу всей выбранной строки, но используются только те, которые находятся в интересующем столбце, а все остальные игнорируются. Таким образом, операция чтения из одной ячейки приводит к разрушению данных всей строки, и их нужно восстанавливать. Регенерация данных после чтения вы-

полняется автоматически интерфейсной логикой микросхемы, и происходит это сразу же после считывания строки.

Теперь рассмотрим различные типы микросхем динамической памяти, начнем с системных DRAM, то есть микросхем, предназначенных для использования в качестве основной памяти. На начальном этапе это были микросхемы асинхронной памяти, работа которых не привязана жестко к тактовым импульсам системной шины.

Асинхронные динамические ОЗУ. Микросхемы асинхронных динамических ОЗУ управляются сигналами RAS и CAS, и их работа в принципе не связана непосредственно тактовыми импульсами шины. Асинхронной памяти свойственны дополнительные затраты времени на взаимодействие микросхем памяти и контроллера. Так, в асинхронной схеме сигнал RAS будет сформирован только после поступления в контроллер тактирующего импульса и будет воспринят микросхемой памяти через некоторое время. После этого память выдаст данные, но контроллер сможет их считать только по приходу следующего тактирующего импульса, так как он должен работать синхронно с остальными устройствами ВМ. Таким образом, на протяжении цикла чтения/записи происходят небольшие задержки из-за ожидания памятью контроллера и контроллером памяти.

Микросхемы DRAM.. В первых микросхемах динамической памяти применялся наиболее простой способ обмена данными, часто называемый традиционным (conventional). Он позволял считывать и записывать строку памяти только на каждый пятый такт (рис. 5.11, *а).* Этапы такой процедуры были описаны ранее. Традиционной DRAM соответствует формула 5-5-5-5. Микросхемы данного типа могли работать на частотах до 40 МГц и из-за своей медлительности (время доступа составляло около 120 нс) просуществовали недолго.

Микросхемы FPM DRAM. Микросхемы динамического ОЗУ, реализующие режим FPM, также относятся к ранним типам DRAM. Сущность режима была показана ранее. Схема чтения для FPM DRAM (рис. 5.11, б) описывается формулой 5-3-3-3 (всего 14 тактов). Применение схемы быстрого страничного доступа позволило сократить время доступа до 60 нс, что, с учетом возможности работать на более высоких частотах шины, привело к увеличению производительности памяти по сравнению с традиционной DRAM приблизительно на 70%. Данный тип микросхем применялся в персональных компьютерах примерно до 1994 года.

Микросхемы EDO DRAM. Следующим этапом в развитии динамических ОЗУ стали ИМС с *гиперстраничным режимом доступа* (HPM, Hyper Page Mode), более известные как EDO (Extended Data Output — расширенное время удержания данных на выходе). Главная особенность технологии — увеличенное по сравнению с FPM DRAM время доступности данных на выходе микросхемы. В микросхемах FPM DRAM выходные данные остаются действительными только при активном сигнале CAS, из-за чего во втором и последующих доступах к строке нужно три такта: такт переключения CAS в активное состояние, такт считывания данных и такт переключения CAS в неактивное состояние. В EDO DRAM по активному (спадающему) фронту сигнала CAS данные запоминаются во внутреннем регистре, где хранятся еще некоторое время после того, как поступит следующий активный фронт сигнала. Это позволяет использовать хранимые данные, когда CAS уже пе-

реведен в неактивное состояние (рис. 5.11, в). Иными словами, временные параметры улучшаются за счет исключения циклов ожидания момента стабилизации данных на выходе микросхемы.

Схема чтения у EDO DRAM уже 5-2-2-2, что на 20% быстрее, чем у FPM. Время доступа составляет порядка 30-40 нс. Следует отметить, что максимальная частота системной шины для микросхем EDO DRAM не должна была превышать 66 МГц.

Микросхемы BEDO DRAM. Технология EDO была усовершенствована компанией VIA Technologies. Новая модификация EDO известна как BEDO (Burst EDO - пакетная EDO). Новизна метода в том, что при первом обращении считывается вся строка микросхемы, в которую входят последовательные слова пакета. За последовательной пересылкой слов (переключением столбцов) автоматически следит внутренний счетчик микросхемы. Это исключает необходимость выдавать адреса для всех ячеек пакета, но требует поддержки со стороны внешней логики. Способ позволяет сократить время считывания второго и последующих слов еще на один такт (рис. 5.11, г), благодаря чему формула приобретает вид 5-1-1-1.

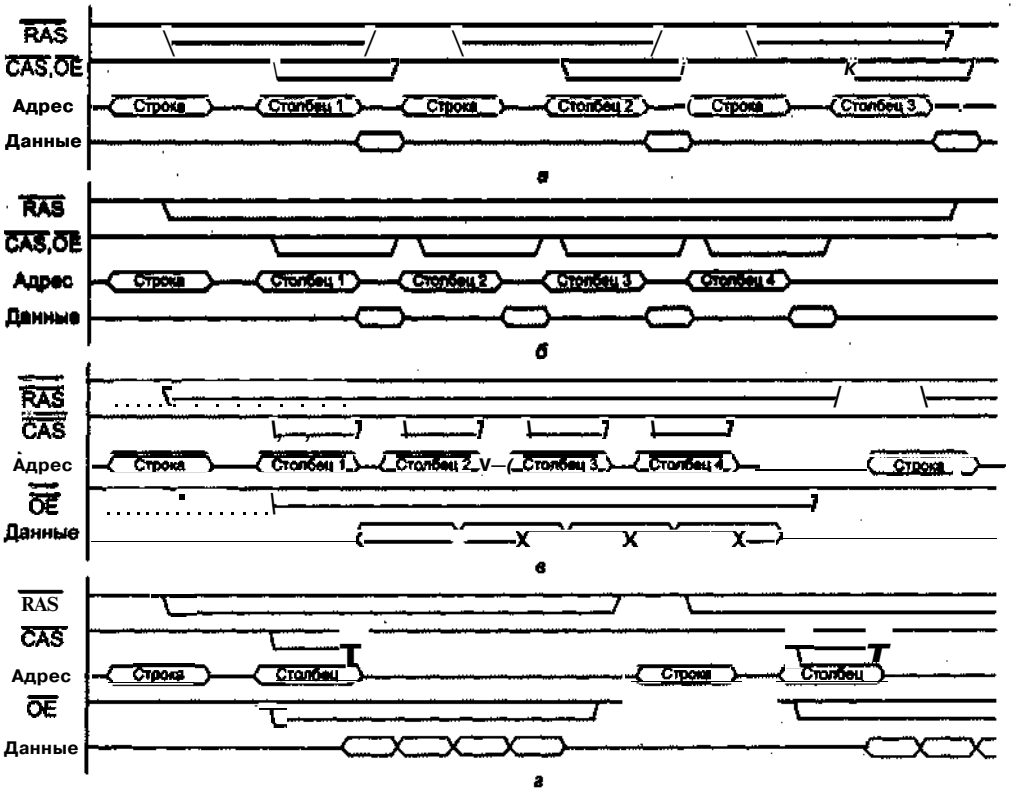


Рис. 5.1 1. Временные диаграммы различных типов асинхронной динамической памяти при длине пакета в четыре слова: а — традиционная DRAM; б — FPM FRAM; в — EDO DRAM; г — BEDO DRAM

Микросхемы EDRAM. Более быстрая версия DRAM была разработана подразделением фирмы Ramtron - компанией Enhanced Memory Systems. Технология реализована в вариантах FPM, EDO и BEDO. У микросхемы более быстрое ядро и внутренняя кэш-память. Наличие последней - главная особенность технологии. В роли кэш-памяти выступает статическая память (SRAM) емкостью 2048 бит. Ядро EDRAM имеет 2048 столбцов, каждый из которых соединен с внутренней кэш-памятью. При обращении к какой-либо ячейке одновременно считывается целая строка (2048 бит). Считанная строка заносится в SRAM, причем перенос информации в кэш-память практически не сказывается на быстродействии, поскольку происходит за один такт. При дальнейших обращениях к ячейкам, относящимся к той же строке, данные берутся из более быстрой кэш-памяти. Следующее обращение к ядру происходит при доступе к ячейке, не расположенной в строке, хранимой в кэш-памяти микросхемы.

Технология наиболее эффективна при последовательном чтении, то есть когда среднее время доступа для микросхемы приближается к значениям, характерным для статической памяти (порядка 10 нс). Главная сложность состоит в несовместимости с контроллерами, используемыми при работе с другими видами DRAM.

Синхронные динамические ОЗУ. В синхронных DRAM обмен информацией синхронизируется внешними тактовыми сигналами и происходит в строго определенные моменты времени, что позволяет взять все от пропускной способности шины «процессор-память» и избежать циклов ожидания. Адресная и управляющая информация фиксируются в ИМС памяти. После чего ответная реакция микросхемы произойдет через четко определенное число тактовых импульсов, и это время процессор может использовать для других действий, не связанных с обращением к памяти. В случае синхронной динамической памяти вместо продолжительности цикла доступа говорят о минимально допустимом периоде тактовой частоты, и речь уже идет о времени порядка 8-10 нс.

Микросхемы SDRAM. Аббревиатура SDRAM (Synchronous DRAM - синхронная DRAM) используется для обозначения микросхем «обычных» синхронных динамических ОЗУ. Кардинальные отличия SDRAM от рассмотренных выше асинхронных динамических ОЗУ можно свести к четырем положениям:

- синхронный метод передачи данных на шину;
- конвейерный механизм пересылки пакета;
- применение нескольких (двух или четырех) внутренних банков памяти;
- передача части функций контроллера памяти логике самой микросхемы.

Синхронность памяти позволяет контроллеру памяти «знать» моменты готовности данных, за счет чего снижаются издержки циклов ожидания и поиска данных. Так как данные появляются на выходе ИМС одновременно с тактовыми импульсами, упрощается взаимодействие памяти с другими устройствами ВМ.

В отличие от BEDO конвейер позволяет передавать данные пакета по тактам, благодаря чему ОЗУ может работать бесперебойно на более высоких частотах, чем асинхронные ОЗУ. Преимущества конвейера особенно возрастают при передаче длинных пакетов, но не превышающих длину строки микросхемы.

Значительный эффект дает разбиение всей совокупности ячеек на независимые внутренние массивы (банки). Это позволяет совмещать доступ к ячейке одного

банка с подготовкой к следующей операции в остальных банках (перезарядкой управляющих цепей и восстановлением информации). Возможность держать открытыми одновременно несколько строк памяти (из разных банков) также способствует повышению быстродействия памяти. При поочередном доступе к банкам частота обращения к каждому из них в отдельности уменьшается пропорционально числу банков и SDRAM может работать на более высоких частотах. Благодаря встроенному счетчику адресов SDRAM, как и BEDO DRAM, позволяет производить чтение и запись в пакетном режиме, причем в SDRAM длина пакета варьируется и в пакетном режиме есть возможность чтения целой строки памяти. ИМС может быть охарактеризована формулой 5-1-1-1. Несмотря на то, что формула для этого типа динамической памяти такая же, что и у BEDO, способность работать на более высоких частотах приводит к тому, что SDRAM с двумя банками при тактовой частоте шины 100 МГц по производительности может почти вдвое превосходить память типа BEDO.

Микросхемы DDR SDRAM. Важным этапом в дальнейшем развитии технологии SDRAM стала DDR SDRAM (Double Data Rate SDRAM - SDRAM с удвоенной скоростью передачи данных). В отличие от SDRAM новая модификация выдает данные в пакетном режиме по обоим фронтам импульса синхронизации, за счет чего пропускная способность возрастает вдвое. Существует несколько спецификаций DDR SDRAM, в зависимости от тактовой частоты системной шины: DDR266, DDR333, DDR400, DDR533. Так, пиковая пропускная способность микросхемы памяти спецификации DDR333 составляет 2,7 Гбайт/с, а для DDR400 - 3,2 Гбайт/с. DDR SDRAM в настоящее время является наиболее распространенным типом динамической памяти персональных ВМ.

Микросхемы RDRAM, DRDRAM. Наиболее очевидные способы повышения эффективности работы процессора с памятью - увеличение тактовой частоты шины либо ширины выборки (количества одновременно пересылаемых разрядов). К сожалению, попытки совмещения обоих вариантов наталкиваются на существенные технические трудности (с повышением частоты усугубляются проблемы электромагнитной совместимости, труднее становится обеспечить одновременность поступления потребителю всех параллельно пересылаемых битов информации). В большинстве синхронных DRAM (SDRAM, DDR) применяется широкая выборка (64 бита) при ограниченной частоте шины.

Принципиально отличный подход к построению DRAM был предложен компанией Rambus в 1997 году. В нем упор сделан на повышение тактовой частоты до 400 МГц при одновременном уменьшении ширины выборки до 16 бит. Новая память известна как RDRAM (Rambus Direct RAM). Существует несколько разновидностей этой технологии: Base, Concurrent и Direct. Во всех тактирование ведется по обоим фронтам синхросигналов (как в DDR), благодаря чему результирующая частота составляет соответственно 500-600, 600-700 и 800 МГц. Два первых варианта практически идентичны, а вот изменения в технологии Direct Rambus (DRDRAM) весьма значительны.

Сначала остановимся на принципиальных моментах технологии RDRAM, ориентируясь в основном на более современный вариант - DRDRAM. Главным отличием от других типов DRAM является оригинальная система обмена данными между ядром и контроллером памяти, в основе которой лежит так называемый

«канал Rambus», применяющий асинхронный блочно-ориентированный протокол. На логическом уровне информация между контроллером и памятью передается пакетами.

Различают три вида пакетов: пакеты данных, пакеты строк и пакеты столбцов. Пакеты строк и столбцов служат для передачи от контроллера памяти команд управления соответственно линиями строк и столбцов массива запоминающих элементов. Эти команды заменяют обычную систему управления микросхемой с помощью сигналов RAS, CAS, WE и CS.

Массив ЗЭ разбит на банки. Их число в кристалле емкостью 64 Мбит составляет 8 независимых или 16 сдвоенных банков. В сдвоенных банках пара банков использует общие усилители чтения/записи. Внутреннее ядро микросхемы имеет 128-разрядную шину данных, что позволяет по каждому адресу столбца передавать 16 байт. При записи можно использовать маску, в которой каждый бит соответствует одному байту пакета. С помощью маски можно указать, сколько байтов пакета и какие именно должны быть записаны в память.

Линии данных, строк и столбцов в канале полностью независимы, поэтому команды строк, команды столбцов и данные могут передаваться одновременно, причем для разных банков микросхемы. Пакеты столбцов включают в себя по два поля и передаются по пяти линиям. Первое поле задает основную операцию записи или чтения. Во втором поле находится либо указание на использование маски записи (собственно маска передается по линиям данных), либо расширенный код операции, определяющий вариант для основной операции. Пакеты строк подразделяются на пакеты активации, отмены, регенерации и команды переключения режимов энергопотребления. Для передачи пакетов строк выделены три линии.

Операция записи может следовать сразу за чтением — нужна лишь задержка на время прохождения сигнала по каналу (от 2,5 до 30 нс в зависимости от длины канала). Чтобы выровнять задержки в передаче отдельных битов передаваемого кода, проводники на плате должны располагаться строго параллельно, иметь одинаковую длину, (длина линий не должна превышать 12 см) и отвечать строгим требованиям, определенным разработчиком.

Каждая запись в канале может быть конвейеризирована, причем время задержки первого пакета данных составляет 50 нс, а остальные операции чтения/записи осуществляются непрерывно (задержка вносится только при смене операции с записи на чтение, и наоборот).

В имеющихся публикациях упоминается работа Intel и Rambus над новой версией RDRAM, названной nDRAM, которая будет поддерживать передачу данных с частотами до 1600 МГц.

Микросхемы SLDRAM. Потенциальным конкурентом RDRAM на роль стандарта архитектуры памяти для будущих персональных ВМ выступает новый вид динамического ОЗУ, разработанный консорциумом производителей ВМ SyncLink Consortium и известный под аббревиатурой SLDRAM. В отличие от RDRAM, технология которой является собственностью компаний Rambus и Intel, данный стандарт — открытый. На системном уровне технологии очень похожи. Данные и команды от контроллера к памяти и обратно в SLDRAM передаются пакетами по 4 или 8 посылок. Команды, адрес и управляющие сигналы посылаются по однонаправленной 10-разрядной командной шине. Считываемые и записываемые данные

передаются по двунаправленной 18-разрядной шине данных. Обе шины работают на одинаковой частоте. Пока что еще эта частота равна 200 МГц, что, благодаря технике DDR, эквивалентно 400 МГц. Следующие поколения SLDRAM должны работать на частотах 400 МГц и выше, то есть обеспечивать эффективную частоту более 800 МГц.

К одному контроллеру можно подключить до 8 микросхем памяти. Чтобы избежать запаздывания сигналов от микросхем, более удаленных от контроллера, временные характеристики для каждой микросхемы определяются и заносятся в ее управляющий регистр при включении питания.

Микросхемы ESDRAM. Это синхронная версия EDRAM, в которой используются те же приемы сокращения времени доступа. Операция записи в отличие от чтения происходит в обход кэш-памяти, что увеличивает производительность ESDRAM при возобновлении чтения из строки, уже находящейся в кэш-памяти. Благодаря наличию в микросхеме двух банков простои из-за подготовки к операциям чтения/записи сводятся к минимуму. Недостатки у рассматриваемой микросхемы те же, что и у EDRAM — усложнение контроллера, так как он должен учитывать возможность подготовки к чтению в кэш-память новой строки ядра. Кроме того, при произвольной последовательности адресов кэш-память задействуется неэффективно.

Микросхемы CDRAM. Данный тип ОЗУ разработан в корпорации Mitsubishi, и его можно рассматривать как пересмотренный вариант ESDRAM, свободный от некоторых ее несовершенств. Изменены емкость кэш-памяти и принцип размещения в ней данных. Емкость одного блока, помещаемого в кэш-память, уменьшена до 128 бит, таким образом, в 16-килобитовом кэше можно одновременно хранить копии из 128 участков памяти, что позволяет эффективнее использовать кэш-память. Замена первого помещенного в кэш участка памяти начинается только после заполнения последнего (128-го) блока. Изменению подверглись и средства доступа. Так, в микросхеме используются раздельные адресные шины для статического кэша и динамического ядра. Перенос данных из динамического ядра в кэш-память совмещен с выдачей данных на шину, поэтому частые, но короткие пересылки не снижают производительности ИМС при считывании из памяти больших объемов информации и уравнивают CDRAM с ESDRAM, а при чтении по выборочным адресам CDRAM явно выигрывает. Необходимо, однако, отметить, что вышеперечисленные изменения привели к еще большему усложнению контроллера памяти.

Постоянные запоминающие устройства

Слово «постоянные» в названии этого вида запоминающих устройств относится к их свойству хранить информацию при отсутствии питающего напряжения. Микросхемы ПЗУ также построены по принципу матричной структуры накопителя, где в узлах расположены перемычки в виде проводников, полупроводниковых диодов или транзисторов, одним концом подключенные к адресной линии, а другим — к разрядной линии считывания. В такой матрице наличие перемычки может означать 1, а ее отсутствие — 0. В некоторых типах ПЗУ элемент, расположенный на перемычке, исполняет роль конденсатора. Тогда заряженное состояние конденсатора означает. 1, а разряженное — 0.

Основным режимом работы ПЗУ является считывание информации, которое мало отличается от аналогичной операции в ОЗУ как по организации, так и по длительности. Именно это обстоятельство подчеркивает общепризнанное название постоянных ЗУ - ROM (Read-Only Memory - память только для чтения). В то же время запись в ПЗУ по сравнению с чтением обычно сложнее и связана с большими затратами времени и энергии. Занесение информации в ПЗУ называют программированием или «прошивкой». Последнее название напоминает о том, что первые ПЗУ выполнялись на базе магнитных сердечников, а данные в них заносились путем прошивки соответствующих сердечников проводниками считывания. Современные ПЗУ реализуются в виде полупроводниковых микросхем, которые по возможностям и способу программирования разделяют на:

- программируемые при изготовлении;
- однократно программируемые после изготовления;
- многократно программируемые.

ПЗУ, программируемые при изготовлении

Эту группу образуют так называемые масочные устройства и именно к ним принято применять аббревиатуру ПЗУ. В литературе более распространено обозначение различных вариантов постоянных ЗУ сокращениями от английских названий, поэтому в дальнейшем будем также использовать аналогичную систему. Для масочных ПЗУ таким обозначением является ROM, совпадающее с общим названием всех типов ПЗУ. Иногда такие микросхемы именуют MROM (Mask Programmable ROM — ПЗУ, программируемые с помощью маски).

Занесение информации в масочные ПЗУ составляет часть производственного процесса и заключается в подключении или не-подключении запоминающего элемента к разрядной линии считывания. В зависимости от этого из ЗЭ будет всегда извлекаться 1 или 0. В роли перемычки выступает транзистор, расположенный на пересечении адресной и разрядной линий. Какие именно ЗЭ должны быть подключены к выходной линии, определяет маска, «закрывающая» определенные участки кристалла. При создании масочных ПЗУ применяются разные технологии. В первом случае маска просто не допускает металлизации участка, соединяющего транзистор с разрядной линией считывания. Вторая технология связана с видом транзистора в узле. Маска определяет, какой полевой транзистор должен быть имплантирован в данный узел - работающий в обогащенном режиме или в режиме обеднения. В третьем варианте маска задает толщину оксидного слоя затвора транзистора. В зависимости от этого на кристалле формируется либо стандартный транзистор, либо транзистор с высоким порогом срабатывания.

В начальный период масочные микросхемы были дороги, однако сейчас это один из наиболее дешевых видов ПЗУ. Для ROM характерна высокая плотность упаковки ЗЭ на кристалле и высокие скорости считывания информации. Основной сферой применения являются устройства, требующие хранения фиксированной информации. Так, подобные ПЗУ часто используют для хранения шрифтов в лазерных принтерах.

Однократно программируемые ПЗУ

Создание масок для ROM оправдано при производстве большого числа копий. Если требуется относительно небольшое количество микросхем с данной информацией, разумной альтернативой являются однократно программируемые ПЗУ.

Микросхемы PROM. В ИМС типа PROM (Programmable ROM - *программируемые ПЗУ)* информация может быть записана только однократно. Первыми такими ПЗУ стали микросхемы памяти на базе плавких предохранителей. В исходной микросхеме во всех узлах адресные линии соединены с разрядными. Занесение информации в PROM производится электрически, путем пережигания отдельных перемычек, и может быть выполнено поставщиком или потребителем спустя какое-то время после изготовления микросхемы. Подобные ПЗУ выпускались в рамках серий К556 и К1556. Позже появились ИМС, где в перемычку входили два диода, соединенные навстречу. В процессе программирования удалить перемычку можно было с помощью электрического пробоя одного из этих диодов. В любом варианте для записи информации требуется специальное оборудование - программаторы. Основными недостатками данного вида ПЗУ были большой процент брака и необходимость специальной термической тренировки после программирования, без которой надежность хранения данных была невысокой.

Микросхемы OTPEPROM. Еще один вид однократно программируемого ПЗУ - это OTP EPROM (One Time Programmable EPROM - EPROM с однократным программированием). В его основе лежит кристалл EEPROM (см. ниже), но помещенный в дешевый непрозрачный пластиковый корпус без кварцевого окна, из-за чего он может быть запрограммирован лишь один раз.

Многократно программируемые ПЗУ

Процедура программирования таких ПЗУ обычно предполагает два этапа: сначала производится стирание содержимого всех или части ячеек, а затем производится запись новой информации.

В этом классе постоянных запоминающих устройств выделяют несколько групп:

- EPROM (Erasable Programmable ROM - стираемые программируемые ПЗУ);
- EEPROM (Electrically Erasable Programmable ROM - электрически стираемые программируемые ПЗУ);
- флэш-память.

Микросхемы EPROM. В EPROM запись информации производится электрическими сигналами, так же как в PROM, однако перед операцией записи содержимое всех ячеек должно быть приведено к одинаковому состоянию (стерто) путем воздействия на микросхему ультрафиолетовым облучением[1]. Кристалл заключен в керамический корпус, имеющий небольшое кварцевое окно, через которое и производится облучение. Чтобы предотвратить случайное стирание информации, после облучения кварцевое окно заклеивают непрозрачной пленкой. Процесс стирания может выполняться многократно. Каждое стирание занимает порядка 20 мин.

[1] Данный вид микросхем иногда обозначают как UV-EPROM (Ultra-Violet EPROM - EPROM, стираемые ультрафиолетовым облучением).

Данные хранятся в виде зарядов плавающих затворов МОП-транзисторов, играющих роль конденсаторов с очень малой утечкой заряда. Заряженный ЗЭ соответствует логическому нулю, а разряженный - логической единице. Программирование микросхемы происходит с использованием технологии инжекции горячих электронов. Цикл программирования занимает нескольких сотен миллисекунд. Время считывания близко к показателям ROM и DRAM.

По сравнению с PROM микросхемы EPROM дороже, но возможность многократного перепрограммирования часто является определяющей. Данный вид ИМС выпускался в рамках серии К573 (зарубежный аналог — серия 27xxx).

Микросхемы EEPROM. Более привлекательным вариантом многократно программируемой памяти является электрически стираемая программируемая постоянная память EEPROM. Стирание и запись информации в эту память производятся побайтово, причем стирание - не отдельный процесс, а лишь этап, происходящий автоматически при записи. Операция записи занимает существенно больше времени, чем считывание — несколько сотен микросекунд на байт. В микросхеме используется тот же принцип хранения информации, что и в EPROM. Программирование EPROM не требует специального программатора и реализуется средствами самой микросхемы.

Выпускаются два варианта микросхем: с последовательным и параллельным доступом, причем на долю первых приходится 90% всех выпускаемых ИМС этого типа. В EEPROM с доступом по последовательному каналу (SEEPROM — Serial EEPROM) адреса, данные и управляющие команды передаются по одному проводу и синхронизируются импульсами на тактовом входе. Преимуществом SEEPROM являются малые габариты и минимальное число линий ввода/вывода, а недостатком — большое время доступа. SEEPROM выпускаются в рамках серий микросхем 24Cxxx, 25Cxxx и 93Cxxx, а параллельные EEPROM — в серии 28Cxxx.

В целом EEPROM дороже, чем EPROM, а микросхемы имеют менее плотную упаковку ячеек, то есть меньшую емкость.

Флэш-память. Относительно новый вид полупроводниковой памяти - это флэш-память (название flash можно перевести как «вспышка молнии», что подчеркивает относительно высокую скорость перепрограммирования). Впервые анонсированная в середине 80-х годов, флэш-память во многом похожа на EEPROM, но использует особую технологию построения запоминающих элементов. Аналогично EEPROM, во флэш-памяти стирание информации производится электрическими сигналами, но не побайтово, а по блокам или полностью. Здесь следует отметить, что существуют микросхемы флэш-памяти с разбивкой на очень мелкие блоки (страницы) и автоматическим постраничным стиранием, что сближает их по возможностям с EEPROM. Как и в случае с EEPROM, микросхемы флэш-памяти выпускаются в вариантах с последовательным и параллельным доступом.

По организации массива ЗЭ различают микросхемы типа:

- Bulk Erase (тотальная очистка) — стирание допустимо только для всего массива ЗЭ;
- Boot Lock — массив разделен на несколько блоков разного размера, содержимое которых может очищаться независимо. У одного из блоков есть аппаратные средства для защиты от стирания;

- Flash File — массив разделен на несколько равноправных блоков одинакового размера, содержимое которых может стираться независимо.

Полностью содержимое флэш-памяти может быть очищено за одну или несколько секунд, что значительно быстрее, чем у EEPROM. Программирование (запись) байта занимает время порядка 10 мкс, а время доступа при чтении составляет 35-200 нс.

Как и в EEPROM, используется только один транзистор на бит, благодаря чему достигается высокая плотность размещения информации на кристалле (на 30% выше чем у **DRAM).**

Наиболее распространенные серии микросхем флэш-памяти - 28Fxxx, 29F/C/EExxx, 39SFxxx (параллельные) и 45Dxxx (последовательные).

Энергонезависимые оперативные запоминающие устройства

Под понятие *энергонезависимое ОЗУ* (NVRAM - Non-Volatile RAM) подпадает несколько типов памяти. От перепрограммируемых постоянных ЗУ их отличает отсутствие этапа стирания, предваряющего запись новой информации, поэтому вместо термина «программирование» для них употребляют стандартный термин «запись».

Микросхемы BBSRAM. К рассматриваемой группе относятся обычные статические ОЗУ со встроенным литиевым аккумулятором и усиленной защитой от искажения информации в момент включения и отключения внешнего питания. Для их обозначения применяют аббревиатуру BBSRAM (Battery-Back SRAM).

Микросхемы NVRAM. Другой подход реализован в микросхеме, разработанной компанией Simtec. Особенность ее в том, что в одном корпусе объединены статическое ОЗУ и перепрограммируемая постоянная память типа EEPROM. При включении питания данные копируются из EEPROM в SRAM, а при выключении — автоматически перезаписываются из SRAM в EEPROM. Благодаря такому приему данный вид памяти можно считать энергонезависимым.

Микросхемы FRAM. FRAM (Ferroelectric RAM - ферроэлектрическая память) разработана компанией Ramtron и представляет собой еще один вариант энергонезависимой памяти. По быстродействию данное ЗУ несколько уступает динамическим ОЗУ и пока рассматривается лишь как альтернатива флэш-памяти. Причисление **FRAM** к оперативным ЗУ обусловлено отсутствием перед записью явно выраженного цикла стирания информации.

Запоминающий элемент FRAM похож на ЗЭ динамического ОЗУ, то есть состоит из конденсатора и транзистора. Отличие заключено в диэлектрических свойствах материала между обкладками конденсатора. В FRAM этот материал (несмотря на название, он не содержит железа и имеет химическую формулу $BaTiO_3$) обладает большой диэлектрической постоянной и может быть поляризован с помощью электрического поля. Поляризация сохраняется вплоть до ее изменения противоположно направленным электрическим полем, что и обеспечивает энергонезависимость данного вида памяти. Данные считываются за счет воздействия на конденсатор электрического поля. Величина возникающего при этом тока зависит от того, изменяет ли приложенное поле направление поляризации на проти-

воположное или нет, что может быть зафиксировано усилителями считывания. В процессе считывания содержимое ЗЭ разрушается и должно быть восстановлено путем повторной записи, то есть как и DRAM, данный тип ЗУ требует регенерации. Количество циклов перезаписи для FRAM обычно составляет 10 млрд.

Главное достоинство данной технологии в значительно более высокой скорости записи по сравнению с EEPROM. В то же время относительная простота ЗЭ позволяет добиться высокой плотности размещения элементов на кристалле, сопоставимой с DRAM. FRAM выпускаются в виде микросхем, полностью совместимых с последовательными и параллельными EEPROM. Примером может служить серия 24Cxx.

Специальные типы оперативной памяти

В ряде практических задач более выгодным оказывается использование специализированных архитектур ОЗУ, где стандартные функции (запись, хранение, считывание) сочетаются с некоторыми дополнительными возможностями или учитывают особенности применения памяти. Такие виды ОЗУ называют специализированными и к ним причисляют:

- память для видеоадаптеров;
- память с множественным доступом (многопортовые ОЗУ);
- память типа очереди (ОЗУ типа FIFO).

Два последних типа относятся к статическим ОЗУ.

Оперативные запоминающие устройства для видеоадаптеров

Использование памяти в видеоадаптерах имеет свою специфику и для реализации дополнительных требований прибегают к несколько иным типам микросхем. Так, при создании динамичных изображений часто достаточно просто изменить расположение уже хранящейся в видеопамяти информации. Вместо того чтобы многократно пересылать по шине одни и те же данные, лишь несколько изменив их расположение, выгоднее заставить микросхему памяти переместить уже хранящиеся в ней данные из одной области ядра в другую. На ИМС памяти можно также возложить операции по изменению цвета точек изображения.

Кратко рассмотрим некоторые из типов ОЗУ, ориентированных на применение в качестве видеопамяти.

Микросхемы CGRAM. Аббревиатура SGRAM (Synchronous Graphic DRAM - синхронное графическое динамическое ОЗУ) обозначает специализированный вид синхронной памяти с повышенной внзгренней скоростью передачи данных. SGRAM может самостоятельно выполнять некоторые операции над видеоданными, в частности блочную запись. Предусмотрены два режима такой записи. В первом - режиме блочной записи (Block Write) - можно изменять цвет сразу восьми элементов изображения (пикселов). Назначение второго режима - блочной записи с маскированием определенных битов (Masked Write или Write-per-Bit) - предотвратить изменение цвета для отдельных пикселов пересылаемого блока. Имеется также модификация данной микросхемы, известная как DDR SGRAM, отличие которой

очевидно из приставки DDR. Использование обоих фронтов синхросигналов ведет к соответствующему повышению быстродействия ИМС.

Микросхемы VRAM. ОЗУ типа VRAM (Video RAM) отличается высокой производительностью и предназначено для мощных графических систем. При разработке ставилась задача обеспечить постоянный поток данных при обновлении изображения на экране. Для типовых значений разрешения и частоты обновления изображения интенсивность потока данных приближается к 200 Мбит/с. В таких условиях процессору трудно получить доступ к видеопамяти для чтения или записи. Чтобы разрешить эту проблему, в микросхеме сделаны существенные архитектурные изменения, позволяющие обособить обмен между процессором и ядром VRAM для чтения/записи информации и операции по выдаче информации на схему формирования видеосигнала (ЦАП - цифро-аналоговый преобразователь). Связь памяти с процессором обеспечивается параллельным портом, а с ЦАП — дополнительным последовательным портом. Кроме того, динамическое ядро DRAM дополнено памятью с последовательным доступом (SAM — Serial Access Memory) емкостью 4 Кбайт. Оба вида памяти связаны между собой широкой внутренней шиной. Выводимая на экран информация порциями по 4 Кбайт из ядра пересылается в SAM и уже оттуда, в последовательном коде (последовательный код формируется с помощью подключенных к SAM сдвиговых регистров), поступает на ЦАП. В момент перезаписи в SAM новой порции ядро VRAM полностью готово к обслуживанию запросов процессора. Наряду с режимами Block Write и Write-per-Bit микросхема реализует режим Flash Write, позволяющий очистить целую строку памяти. Имеется также возможность маскировать определенные ячейки, защищая их от записи.

Микросхемы WRAM. Данный вид микросхем, разработанный компанией Samsung, во многом похож на VRAM. Это также двухпортовая память, допускающая одновременный доступ со стороны процессора и ЦАП, но по конструкции она несколько проще, чем VRAM. Имеющиеся в VRAM, но редко используемые функции исключены, а вместо них введены дополнительные функции, ускоряющие вывод на экран текста и заполнение одним цветом больших площадей экрана. В WRAM применена более быстрая схема буферизации данных и увеличена разрядность внутренней шины. Ускорено также ядро микросхемы, за счет использования режима скоростного страничного режима (UFP — Ultra Fast Page), что обеспечивает время доступа порядка 15 нс. В среднем WRAM на 50% производительнее, чем VRAM, и на 20% дешевле. Применяется микросхема в мощных видеоадаптерах.

Микросхемы MDRAM. Микросхема типа MDRAM (Multibank DRAM - многоблочное динамическое ОЗУ) разработана компанией MoSys и ориентирована на графические карты. Память содержит множество независимых банков по 1K 32-разрядных слов каждый. Банки подключены к быстрой и широкой внутренней шине. Каждый банк может выполнять определенные операции независимо от других банков. Отказ любого из банков ведет лишь к сокращению суммарной емкости памяти и некоторому снижению показателей быстродействия. Благодаря блочному построению технология позволяет изготавливать микросхемы практически любой емкости, не обязательно кратной степени числа 2.

Микросхемы 3D-RAM. Этот тип памяти разработан совместно компаниями Mitsubishi и Sun Microsystems с ориентацией на трехмерные графические ускорители. Помимо массива запоминающих элементов, микросхема 3D-RAM (трехмерная RAM) содержит процессор (арифметико-логическое устройство) и кэш-память. Процессор позволяет выполнять некоторые операции с изображением прямо в памяти. Основные преобразования над пикселами реализуются за один такт, поскольку стандартная последовательность действий «считал, изменил, записал» сводится к одной операции — «изменить», выполняемой в момент записи. Процессор микросхемы позволяет за секунду выполнить около 400 млн операций по обработке данных и закрасить до 4 млн элементарных треугольников. Кэш-память обеспечивает более равномерную нагрузку на процессор при интенсивных вычислениях. Ядро 3D-RAM состоит из четырех банков общей емкостью 10 Мбит. Размер строк памяти выбран таким, чтобы в пределах одной и той же области памяти находилось как можно больше трехмерных объектов. Это дает возможность сэкономить время на переходы со строки на строку. По цене данный тип микросхем сравним с VRAM.

Многопортовые ОЗУ

Стандартное однопортовое ОЗУ имеет по одной шине адреса, данных и управления и в каждый момент времени обеспечивает доступ к ячейке памяти только одному устройству. Структура запоминающего элемента (ЗЭ) такого ОЗУ приведена на рис. 5.12, *а*.

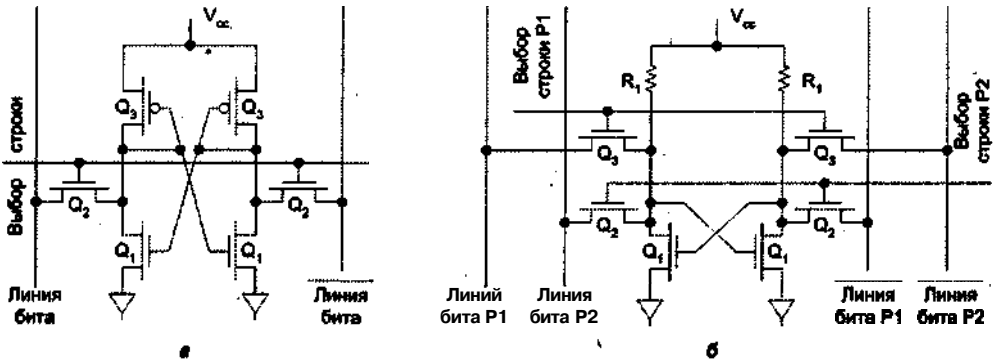


Рис. 5.12. Запоминающие элементы статического ОЗУ: а - однопортового; *б* - двухпортового

В отличие от стандартного в *n*-портовом ОЗУ имеется *n* независимых наборов шин адреса, данных и управления, гарантирующих одновременный и независимый доступ к ОЗУ *n* устройствам. Данное свойство позволяет существенно упростить создание многопроцессорных и многомашинных вычислительных систем, где многопортовое ОЗУ выступает в роли общей или совместно используемой памяти. В рамках одной ВМ подобное ОЗУ может обеспечивать обмен информацией между ЦП и УВВ (например, контроллером магнитного диска) намного эффективней, чем прямой доступ к памяти. В настоящее время серийно выпускаются

двух- и четырехпортовые микросхемы, среди которых наиболее распространены первые. Поскольку архитектурные решения в обоих случаях схожи, дальнейшее изложение будет вестись применительно к двухпортовым ОЗУ.

ЗЭ двухпортового ОЗУ (см. рис. 5.12, *б)* также содержит шесть транзисторов, но в отличие от стандартного ЗЭ (см. рис. 5.12, *а)* транзисторы Q3 служат не в качестве резисторов, а предоставляют доступ к элементу с двух направлений.

В двухпортовой памяти имеются два набора адресных, информационных и управляющих сигнальных шин, каждый из которых обеспечивает доступ к общему массиву ЗЭ (рис. 5.13). Поскольку двухпортовому ОЗУ свойственна симметричная структура, в дальнейшем наборы шин будем называть «левым» (Л) и «правым» (П). В целом организация матрицы ЗЭ остается традиционной.

Доступ к ячейкам возможен как через левую, так и через правую группу шин, причем если Л- и П-адреса различны, никаких конфликтов не возникает. Проблемы потенциально возможны, когда Л- и П-устройства одновременно обращаются по одному и тому же адресу и хотя бы одно из этих устройств пытается выполнить операцию записи. В этом случае, если один из портов читает информацию, а другой производит запись в ту же ячейку, вероятно считывание недостоверной информации. При попытке единовременного ввода в ячейку с двух направлений в нее может быть занесена неопределенная комбинация из записываемых слов. Несмотря на то что вероятность подобных ситуаций по оценкам не превышает 0,1%, такой вариант необходимо учитывать, для чего в двухпортовой памяти имеется схема арбитража с использованием сигналов «Занято».

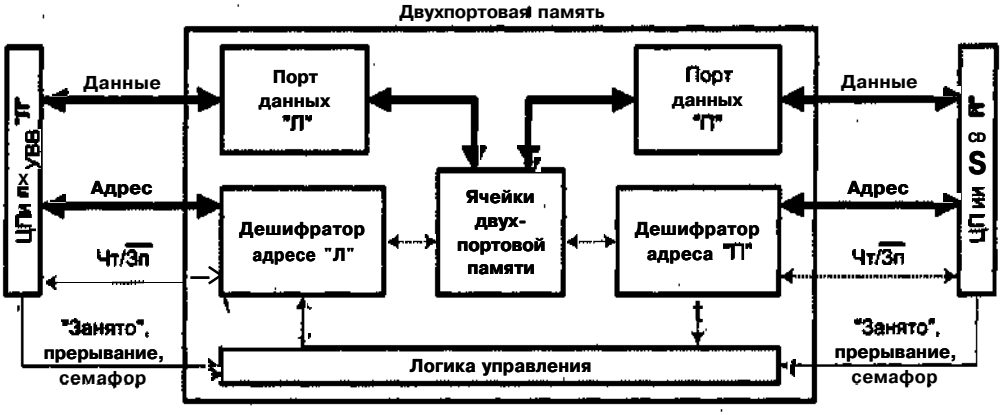


Рис. 5.13. Структура двухпортового ОЗУ

Логика арбитража в микросхеме реализована аппаратными средствами (рис. 5.14).

Схема обеспечивает формирование сигнала «Занято», запрещающего запись в ячейку для той половины, на которой адрес появится позже, а также принятие решения в пользу одного из входных портов при одновременном поступлении адресов. Арбитр содержит два компаратора адресов (КЛ и КП), два буфера задержки (БЗЛ и БЗП), триггер-защелку (ТЗ), образованный перекрестно связанными схемами «И-НЕ», и формирователи сигналов «Занято» (ЗЛ и ЗП).

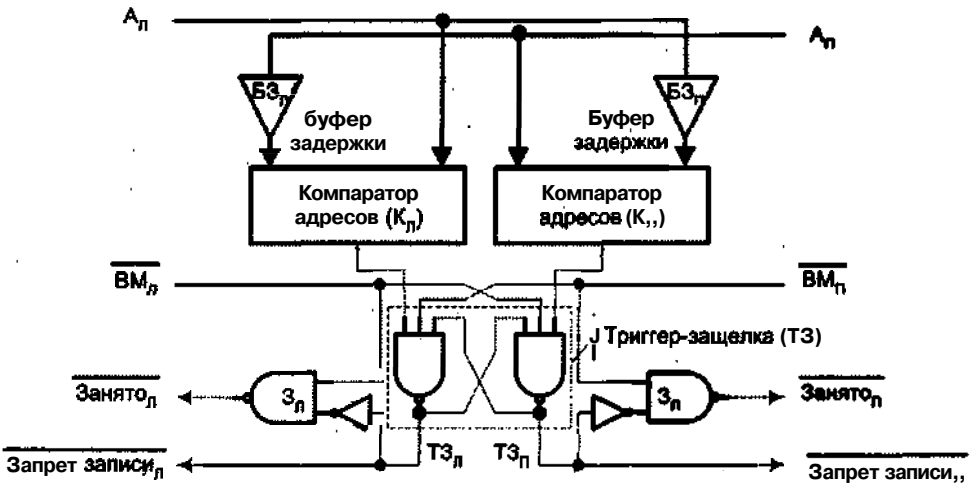


Рис. 5.14. Логика арбитража с использованием сигнала «Занято»

Выявление адреса, поступившего первым, производится за счет буферов задержки и компараторов адресов. Так, если информация на адресной шине $A_л$ уже стабилизировалась, а на шине $A_п$ имеет тенденцию совпасть с $A_л$, то на выходе $K_п$ сигнал «l» появится немедленно, как только адреса совпадут, в то время как на выходе $K_л$ он сформируется с некоторой задержкой, определяемой $БЗ_л$. Эта ситуация фиксируется триггером-защелкой, на выходе которого возможны три комбинации сигналов: $ТЗ_л = ТЗ_п = 1$, $ТЗ_л = 0$ и $ТЗ_п = 1$, $ТЗ_л = 1$ и $ТЗ_п = 0$. В исходном состоянии $ТЗ_л = ТЗ_п = 1$, поскольку на выходах обоих компараторов 0. В нашем случае при поступлении от $K_п$ сигнала 1 на выходе $ТЗ_п$ установится значение G, в то время как выход $ТЗ_л$ по-прежнему останется в состоянии 1, так как с $K_л$ единица придет позже. В результате будет сформирован сигнал, запрещающий запись через правый порт, а также сигнал $Занято_п$, который поступает на устройство, подключенное к правому порту микросхемы, и может быть использован для задержки или повторения обращения к ячейке.

Если обращения к одной и той же ячейке происходят строго одновременно, сигналы с выходов $K_л$ и $K_п$ поступят на входы ТЗ также одновременно. Схема ТЗ реализована таким образом, что триггер и в этом случае установится в какое-то одно из двух своих нормальных состояний, что гарантирует принятие положительного решения на доступ к ячейке в пользу только одного из портов.

Сигналы выбора микросхемы $ВМ_л$ и $ВМ_п$ поступают непосредственно на ТЗ, благодаря чему при наличии обращения только от одного из портов арбитраж не производится.

Помимо возможности доступа к ячейкам с двух направлений, двухпортовая память снабжается средствами для обмена сообщениями между подключенными к ней устройствами: системой прерывания и системой семафоров. Первую из них называют аппаратной, а вторую - программной.

В системе прерываний двухпортовой памяти две последних ячейки микросхемы (с наибольшими адресами) используются в качестве «почтовых ящиков» для

обмена сообщениями между устройствами, подключенными к Л- и П-портам. Сообщению от левого устройства выделена ячейка с четным адресом (если емкость памяти равна 1K, то это будет адрес $3FF_{16}$), а от правого — с нечетным ($3FE_{16}$). Когда устройство записывает информацию в свой «почтовый ящик», формируется запрос прерывания к устройству, подключенному к противоположному порту. Этот сигнал автоматически сбрасывается, когда адресат считывает информацию из своего «почтового ящика».

Система семафоров - это имеющийся в двухпортовой памяти набор из восьми триггеров, состояние которых может быть прочитано и изменено со стороны любого из портов. Триггеры играют роль программных семафоров или флагов, с помощью которых Л- и П-устройства могут извещать друг друга о каких-то событиях. Сущность этих событий не зафиксирована и определяется реализуемыми программами. Обычно семафоры нужны для предоставления одному из процессоров монопольного права работы с определенным блоком данных до завершения всех необходимых операций с этим блоком. В этом случае процессор, монополизирующий блок данных, устанавливает один из семафоров в состояние 1, а по завершении — в 0. Второй процессор, прежде чем обратиться к данному блоку, считывает семафор и при единичном состоянии последнего повторяет считывание и анализ семафора до тех пор, пока первый процессор не установит его в состояние 0. Естественно, что в программном обеспечении Л- и П-процессоров распределение и правила использования семафоров должны быть согласованы.

Зачастую одной микросхемы многопортовой памяти не хватает из-за недостаточной емкости одной ИМС или ввиду малой разрядности ячеек. В обоих случаях необходимо соединить несколько микросхем, соответственно параллельно или последовательно. Если несколько микросхем объединяются в цепочку для достижения нужной разрядности слова, возникает проблема с арбитражем при одновременном обращении к одной и той же ячейке. В этих случаях в разных ИМС цепочки, в силу разброса их параметров, предпочтение может быть отдано разным портам, в то время как решение должно быть единым. Для исключения подобной ситуации микросхемы многопортовой памяти выпускаются в двух вариантах: ведущие (master) и ведомые (slave). Принятие решения производится только в ведущих микросхемах, а ведомые функционируют в соответствии с инструкцией, полученной от ведущего. Таким образом, в цепочке используется только одна микросхема типа «ведущий», а все прочие ИМС должны иметь тип «ведомый».

Память типа FIFO

Во многих случаях ОЗУ применяется для буферизации потока данных, когда данные считываются из памяти в той же последовательности, в которой они туда заносились, но поступление и считывание происходят с различной скоростью. Часто для этой цели применяют обычное ОЗУ, однако здесь одновременная запись и считывание информации невозможны. Более эффективным видом ОЗУ, где оба действия могут вестись одновременно, служит память типа FIFO. Микросхема представляет собой двухпортовое ОЗУ, где один порт предназначен для занесения информации, а второй - для считывания. Для FIFO-памяти характерны все технологические приемы, свойственные двухпортовой памяти, в частности спосо-

бы арбитража при одномоментном обращении к одной и той же ячейке. В то же время есть и существенные отличия.

Первое состоит в том, что у микросхемы нет входов для указания адреса ячейки, занесение и считывание данных производится в порядке их поступления через одну входную точку и одну выходную.

Второе отличие связано с необходимостью слежения за состоянием очереди. Для этого в микросхеме имеются регистры-указатели адресов начала и конца очереди, а также специальные флаги, которые указывают на две ситуации: отсутствие данных (в этом случае блокируется считывание из микросхемы) и полное заполнение памяти (блокируется запись).

Обнаружение и исправление ошибок

При работе с полупроводниковой памятью не исключено возникновение различного рода отказов и сбоев. Причиной *отказов* могут быть производственные дефекты, повреждение микросхем или их физический износ. Проявляются отказы в том, что в отдельных разрядах одной или нескольких ячеек постоянно считывается 0 или 1, вне зависимости от реально записанной туда информации. *Сбой* — это случайное событие, выражающееся в неверном считывании или записи информации в отдельных разрядах одной или нескольких ячеек, не связанное с дефектами микросхемы. Сбои обычно обусловлены проблемами с источником питания или с воздействием альфа-частиц, возникающих в результате распада радиоактивных элементов, которые в небольших количествах присутствуют практически в любых материалах. Как отказы, так и сбои крайне нежелательны, поэтому в большинстве систем основной памяти содержатся схемы, служащие для обнаружения и исправления ошибок.

Вне зависимости от того, как именно реализуется контроль и исправление ошибок, в основе их всегда лежит введение избыточности. Это означает, что контролируемые разряды дополняются контрольными разрядами, благодаря которым и возможно детектирование-ошибок, а в ряде методов — их коррекция. Общую схему обнаружения и исправления ошибок иллюстрирует рис. 5.15.

На рисунке показано, каким образом осуществляются обнаружение и исправление ошибок. Перед записью М-разрядных данных в память производится их обработка, обозначенная на схеме функцией «f», в результате которой формируется добавочный *K*-разрядный код. В память заносятся как данные, так и этот вычисленный код, то есть $(M + K)$-разрядная информация. При чтении информации повторно формируется *K*-разрядный код, который сравнивается с аналогичным кодом, считанным из ячейки. Сравнение приводит к одному из трех результатов:

- Не обнаружено ни одной ошибки. Извлеченные из ячейки данные подаются на выход памяти.
- Обнаружена ошибка, и она может **быть исправлена.** Биты данных и добавочного кода подаются на схему коррекции. После исправления ошибки в М-разрядных данных они поступают на выход памяти.
- Обнаружена ошибка, и она не **может быть исправлена.** Выдается сообщение о неисправимой ошибке.

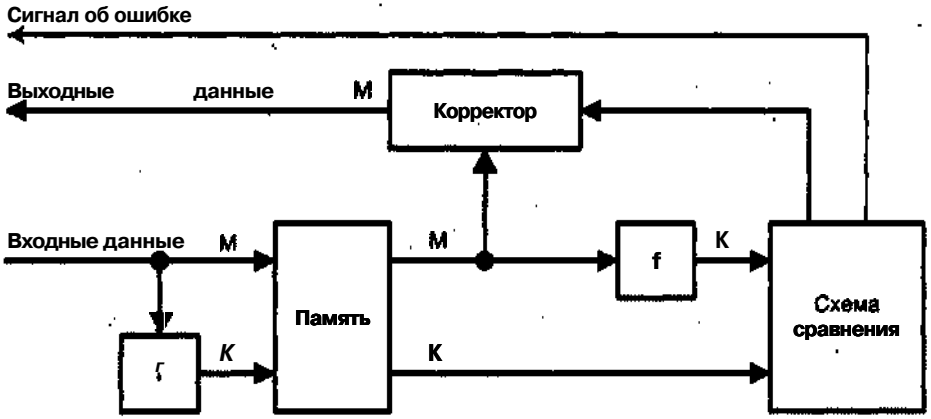


Рис. 5.15 Общая схема обнаружения и исправления ошибок [200]

Коды, используемые для подобных операций, называют *корректирующими кодами* или *кодами с исправлением ошибок*.

Простейший вид такого кода основан на добавлении к каждому байту информации одного *бита паритета*. Бит паритета - это дополнительный бит, значение которого устанавливается таким, чтобы суммарное число единиц в данных, с учетом этого дополнительного разряда, было четным (или нечетным). В ряде систем за основу берется четность, в иных - нечетность. Для 64-разрядного слова требуется восемь битов паритета, то есть ячейка памяти должна хранить 36 разрядов. При записи слова в память для каждого байта формируется бит паритета. Это может быть сделано с помощью схемы в виде дерева, составленного из схем сложения по модулю 2. При чтении из памяти выполняется аналогичная операция над считанными информационными битами, а ее результат сравнивается с битом паритета, вычисленным при записи и хранившимся в памяти. Метод позволяет обнаружить ошибку, если исказилось нечетное количество битов. При четном числе ошибок метод неработоспособен. К сожалению, фиксируя ошибку, данный способ кодирования не может указать на ее местоположение, что позволило бы внести исправления, в силу чего его называют *кодом с обнаружением ошибки* (EDC - Error Detection Code).

В основе корректирующих кодов лежит достаточно простая идея [39]. Для контроля двоичного информационного кода длиной M бит добавим к ней K дополнительных контрольных разрядов так, что общая длина последовательности теперь будет равна $M + K$ разрядам. В этом случае из возможных $N = 2^{M+K}$ комбинаций интерес представляют только $L = 2^{M}$ последовательностей, которые называют *разрешенными*. Оставшиеся $N - L$ последовательностей назовем *запрещенными*. Если при обработке (записи в память, считывании или передаче) разрешенной кодовой последовательности произойдут ошибки и возникнет одна из запрещенных последовательностей, то тем самым эти ошибки обнаруживаются. Если же ошибки превратят одну разрешенную последовательность в другую, то такие ошибки не могут быть обнаружены. Для исправления ошибок необходимо произвести разбиение множества запрещенных последовательностей на L непересекающихся подмно-

жеств и каждому подмножеству поставить в соответствие одну из разрешенных последовательностей. Тогда, если была принята некоторая запрещенная последовательность, входящая в одно из подмножеств, считается, что передана разрешенная последовательность, соответствующая этому подмножеству, производится замена, чем и исправляется возникшая ошибка.

Простейший вариант корректирующего кода также может быть построен на базе битов паритета. Для этого биты данных представляются в виде матрицы, к каждой строке и столбцу которой добавляется бит паритета. Для 64-разрядных данных этот подход иллюстрирует табл. 5.1 [200]. Здесь *D* - биты данных, *C*- столбец битов паритета строк, *K* - строка битов паритета столбцов, *P* — бит паритета, контролирующий столбец *C* и строку *K*. Таким образом, к 64 битам данных нужно добавить 17 бит паритета: по 8 бит на строки и столбцы и один дополнительный бит для контроля строки и столбца битов паритета. Если в одной строке и одном столбце обнаружено нарушение паритета, для исправления ошибки достаточно просто инвертировать бит на пересечении этих строки и столбца. Если ошибка паритета выявлена только в одной строке или только одном столбце либо одновременно в нескольких строках и столбцах, фиксируется многобитовая ошибка и формируется признак невозможности коррекции.

Таблица 5.1. Формирование корректирующего кода для 64-битовых данных

	0	1	2	3	4	5	в	7	
0	A	**A**	A	A	A	A	A	A	C_0
1	A	A	D_{10}	A.	D_{12}	D_{13}	D_{14}	D_{15}	C,
2	D_{16}	D_{17}	D_{18}	D_{19}	D_{20}	A.	D_{22}	Аз	C_2
3	D_{24}	D_{25}	D_{26}	D_{27}	A*	A»	АО	At	C_3
4	D_{32}	Аз	Ач	D_{35}	D_{36}	D_{37}	A»	D_{39}	C_4
6	At	A,	D_{42}	Аз	A.	D_{45}	D_{46}	A,	C_5
в	**As**	D_{49}	D_{50}	A,	D_{52}	D_{53}	A.	D_{55}	C_6
7	A.	D_{57}	A*	A*	D_{60}	A.	D_{62}	Аз	C_7
	K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7	P

Недостаток рассмотренного приема в том, что он требует большого числа дополнительных разрядов. Более эффективным представляется код, предложенный Ричардом Хэммингом и носящий его имя *(код Хэмминга)*. Логику этого кода для четырехразрядных слов *(M = 4)* иллюстрирует диаграмма Венна[1], приведенная на рис. 5.16

Три пересекающихся окружности образуют семь сегментов. Четырем битам данных назначаются внутренние сегменты (см. рис. 5.16, *а),* а остальные сегменты заполняются битами паритета. Биты паритета выбираются таким образом, чтобы

[1] Диаграмма Венна - это графическое представление операций над множествами, где множества обозначаются замкнутыми областями, содержащими внутри себя все элементы этих множеств. Диаграмма используется для отображения логических отношений между элементами множеств. Так, заштрихованная область на рис. 5.16, *г* есть представление функции пересечения множеств А, Б и В (математически это записывается как **А *п* Б ∩В).**

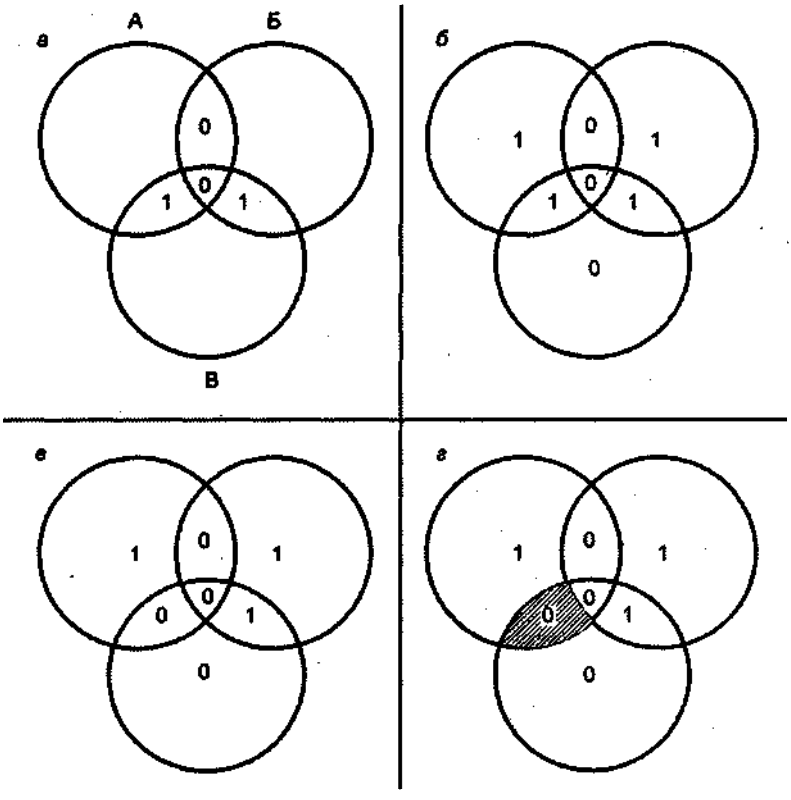


Рис. 5.16. Иллюстрация кода Хэмминга для четырехразрядного слова

общее число единиц в каждой окружности было четным (см. рис. 5.16, б). Так, поскольку окружность А содержит одну единицу, бит паритета для нее принимается равным 1. Теперь, если в результате ошибки изменится один из битов данных (см. рис. 5.16, *в),* это легко выявить. Путем проверки паритета обнаруживаются несоответствия в окружностях А и В. Для окружности Б несоответствия нет. Только один из семи сегментов присутствует в окружностях А и В и отсутствует в Б, и ошибка может быть исправлена за счет изменения бита в этом сегменте (см. рис. 5.16, г).

Для пояснения концепции, положенной в основу кода Хэмминга, построим код, обнаруживающий и исправляющий однобитовые ошибки в 8-разрядных словах (пример взят из [47]).

Сначала определим требуемую длину корректирующего кода. В соответствии с рис. 5.15, на вход схемы сравнения поступают два *К*-разрядных значения. Сравнение производится путем поразрядной операции «исключающее ИЛИ» (сложение по модулю 2) над входными кодами. Результатом является так называемое *слово синдрома.* В зависимости от того, было ли совпадение входных кодов или нет, соответствующий бит синдрома будет равен 0 или 1.

Слово синдрома состоит из *К* разрядов, то есть его возможные значения лежат в диапазоне от 0 до 2^K - 1. Значение 0 соответствует случаю, когда ошибки не обнаружено, остальные 2^K - *1* случая свидетельствуют о наличии ошибки и указывают

на ее местоположение. Поскольку ошибка может возникнуть в любом из M битов данных или K контрольных битов, мы должны иметь $2^K - 1 > M + K$. Это выражение позволяет определить число битов, необходимое для исправления одиночной ошибки в M-разрядных данных.

В табл. 5.2 приведено количество корректирующих разрядов, нужное для контроля данных различной разрядности. Из таблицы видно, что для 8-разрядных слов требуется четыре корректирующих разряда.

Для удобства будем формировать четырехразрядный синдром со следующими характеристиками:

- Если синдром содержит все нули, значит, не обнаружено ни одной ошибки.
- Если синдром содержит единственную единицу в одном из разрядов, это означает, что выявлена ошибка в одном из четырех корректирующих разрядов и никакой коррекции не требуется.
- Если в синдроме в единичное состояние установлены несколько битов, то численное значение синдрома соответствует позиции ошибки в данных, для исправления которой необходимо инвертировать бит в этой позиции.

Таблица 5.2. Количество корректирующих разрядов, используемое в коде Хэмминга

Разрядность	Исправление одиночной ошибки		Исправление одиночной ошибки Обнаружение двойной ошибки	
	Контрольные биты	% увеличения длины кода	Контрольные биты	% увеличения длины кода
8	4	50	5	62,5
16	5	31,25	6	37,5
32	6	18,75	7	21,875
64	7	10,94	8	12,5
128	8	6,25	9	7,03
256	9	3,52	10	3,91

Под контрольные разряды отводятся те биты, чьи позиционные номера представляютсобой степень числа 2 (табл. 5.3). Отдельный контрольный разряд отвечает за определенные биты данных. Так, разрядная позиция n контролируется теми битами Pi , которые делают справедливым соотношение $\Sigma_i = n$. Например, разряд данных с позиционным номером 7_{10} (0111_2) контролируется битами 4, 2 и 1 (7 - 4 + 2 + 1), а разряд с номером 10_{10} (1010_2) - битами 8 и 2 (10 - 8 + 2).

Таблица 5.3. Распределение разрядов 12-разрядного слова между данными и контрольным кодом

Десятичный номер позиции	12	11	10	9	8	7	6	5	4	3	2	1
Двоичный код номера позиции	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001

Биты данных	A	D_7	A	A		A	A	D_2		A		
Контрольные биты					P_8				P_4		P_2	P_1

Таким образом, для рассматриваемого случая значения контрольных разрядов вычисляются по следующим правилам:

$$P_1 = D_1 \oplus D_2 \oplus D_4 \oplus D_5 \oplus A;$$
$$P_2 = D_1 \oplus D_3 \oplus D_4 \oplus D_6 \oplus A;$$
$$P_4 = D_2 \oplus D_3 \oplus D_4 \oplus D_8;$$
$$P_8 = D_5 \oplus D_6 \oplus D_7 \oplus D_8.$$

Проверим корректность такой схемы на примере. Пусть входной крд равен 00101011, где разряду Z), соответствует правая цифра. Контрольные разряды вычисляются следующим образом:

$$P_1 = D_1 \oplus D_2 \oplus D_4 \oplus D_5 \oplus D_7 = 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 = 1;$$
$$P_2 = D_1 \oplus D_3 \oplus D_4 \oplus D_6 \oplus D_7 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1;$$
$$P_4 = D_2 \oplus D_3 \oplus D_4 \oplus D_8 = 1 \oplus 0 \oplus 1 \oplus 0 = 0;$$
$$P_8 = D_5 \oplus D_6 \oplus D_7 \oplus D_8 = 0 \oplus 1 \oplus 0 \oplus 0 = 1.$$

Предположим, что данные в бите 3 содержат ошибку и там вместо 0 находится 1. После пересчета контрольных разрядов имеем

$$P_1' = D_1 \oplus D_2 \oplus D_4 \oplus D_5 \oplus D_7 = 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 = 1;$$
$$P_2' = D_1 \oplus D_3 \oplus D_4 \oplus D_6 \oplus D_7 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0;$$
$$P_4' = D_2 \oplus D_3 \oplus D_4 \oplus D_8 = 1 \oplus 1 \oplus 1 \oplus 0 = 1;$$
$$P_8' = D_5 \oplus D_6 \oplus D_7 \oplus D_8 = 0 \oplus 1 \oplus 0 \oplus 0 = 1.$$

Путем сложения по модулю двух результатов старой и новой проверок получим слово синдрома $(S_8S_4S_2S_1)$:

$$S_1 = P_1 \oplus P_1' = 1 \oplus 1 = 0;$$
$$S_2 = P_2 \oplus P_2' = 1 \oplus 1 = 0;$$
$$S_4 = P_4 \oplus P_4' = 1 \oplus 1 = 0;$$
$$S_8 = P_8 \oplus P_8' = 1 \oplus 1 = 0.$$

Результат 0110_2 (6_{10}) означает, что в разряде 6, содержащем третий бит данных, присутствует ошибка.

Описанный код называется *кодом с исправлением одиночной ошибки* (SEC — Single Error Correcting). В большинстве микросхем памяти используется *код с исправлением одиночной и обнаружением двойной ошибки (SECDED* - Single Error Correcting, Double Error Detecting). Из табл. 5.2 видно, что, по сравнению с SEC, такой код требует одного дополнительного контрольного разряда.

Коды с исправлением ошибок применяются в большинстве ВМ. Например, в основной памяти ВМ типа IBM 3Oxx используется 8-разрядный код SECDED на каждые 64 бита данных, то есть емкость памяти примерно на 12% больше, чем имеет в своем распоряжении пользователь. В ВМ типа VAX на каждые 32 разряда данных добавлен 7-разрядный код SECDED, следовательно, избыточность составляет 22%.

Структура одного из вариантов построения устройства для обнаружения одинарных и двойных ошибок с коррекцией одинарных ошибок приведена на рис. 5.17. Схема предназначена для контроля 16-разрядных данных и размещается между процессором и памятью. Из табл. 5.2 видно, что код SECDED предполагает шесть дополнительных разрядов. Таким образом, из процессора и в процессор поступают 16-разрядные коды *(UD_{15}... UD),* а в память заносятся 22-разрядные данные (M_{21}... M_0). Хранящаяся в ячейках памяти информация состоит из 16 бит информации *(MD_{15}... MD_0)* и 6 контрольных битов *(MP_5... MP).* В последующем первые буквы в обозначении разрядов могут быть опущены, при этом *D* будет означать информационный разряд кода, а *P* - контрольный разряд.

Система размещения основных и контрольных разрядов была рассмотрена ранее, и для данной схемы она приведена в табл. 5.4.

Таблица 5.4. Распределение информационных и контрольных разрядов в 22-разрядном слове

Номер позиции	22	21	20	19	18	17	16	15	14	13	12
Биты данных		D_{15}	D_{14}	D_{13}	D_{12}	A.		D_{10}	D_9	A	A
Контрольные биты	P_5						ft				
Номер позиции	11	10	9	8	7	6	5	4	3	2	1
Биты данных	A	A	A		A	A		A	A		
Контрольные биты				P_3				ft		Л	ft

Контрольные разряды P_4... P_0 вычисляются по стандартной схеме **кода Хэмминга:**

$$P_0 = D_{15} \oplus D_{13} \oplus D_{11} \oplus D_{10} \oplus D_8 \oplus D_6 \oplus D_4 \oplus D_3 \oplus D_1 \oplus D_0;$$

$$P_1 - D_{13} \Phi D_{12} \oplus D_{10} \oplus D_9 \Phi D_6 \copyright A \copyright D_3 \Phi D_2 \copyright D_0;$$

$$P_2 - D_{15} \Phi D_{14} \copyright D_{10} \oplus D_9 \copyright D_8 \circledR D_7 \oplus D_3 \oplus D_2 \Phi A;$$

$$P_3 = D_{10} \oplus D_9 \oplus D_8 \oplus D_7 \oplus A \copyright A \circledR A;$$

$$P_4 - D_{15} \oplus D_{14} \oplus D_{13} \oplus D_{12} \oplus At\text{-}$$

Особенность рассматриваемой схемы состоит в способе формирования контрольного разряда P_5. Он вычисляется путем суммирования по модулю 2 всех остальных 21 разрядов кода (D_{15}... D_0 и MP_4... MP). По мнению авторов рассматриваемой схемы, это облегчает фиксацию факта неисправимой двойной ошибки.

При чтении из памяти формирователем синдрома вычисляется синдром S_4... S_0:

$$S_0 = P_0 \oplus MP_0;$$
$$S_1 = P_1 \oplus MP_1;$$
$$S_2 = P_2 \oplus MP_2;$$
$$S_3 = P_3 \oplus MP_3;$$
$$S_4 = P_4 \oplus MP_4.$$

Здесь P_4... P_0 — контрольные разряды, вычисленные генератором контрольных разрядов на основании информационных битов считанного из памяти кода *(MD_{15}... MD). MP_5... MP_0* — такие же контрольные разряды, полученные тем же генератором, но перед записью информации в память и хранившиеся там вместе с основ-

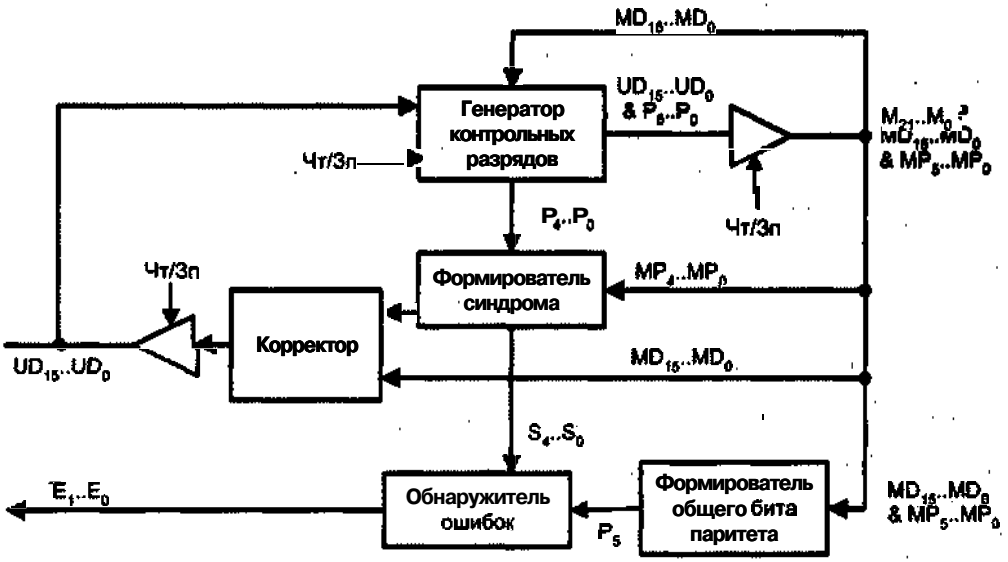


Рис. 5.17. Схема обнаружения одинарных и двойных ошибок с коррекцией одинарных

ной информацией. Если синдром не равен нулю, то он указывает на позицию исказившегося бита при одиночной ошибке. С учетом контрольного разряда P_5 возможны четыре ситуации, показанные в табл. 5.5.

Таблица 5.5. Обнаружение ошибок

Синдром	P5	Тип ошибки	Примечания
0	0	Ошибки нет	
≠0	1	Одиночная ошибка	Корректируемая: синдром указывает на позицию искаженного разряда
≠0	0	Двойная ошибка	Неисправимая
0	1	Ошибка в контрольном разряде	Искажен контрольный разряд P_5, и он может быть откорректирован

Сигнал Чт/Зп на схеме определяет выполняемую операцию. Чт/Зп = 1 означает чтение из памяти, а Чт/Зп = 0 — запись в память.

Проанализировав полученный синдром и разряд P, обнаружитель ошибки формирует двухразрядный код ошибки E_1E_0(табл. 5.6).

Таблица 5.6. Кодирование ошибок

Код ошибки (E1, EO)	Описание
00	Ошибки нет
01	Одиночная ошибка - исправимая
10	Двойная ошибка - неисправимая
И	Ошибка в контрольном разряде — исправимая

Стековая память

Стековая память обеспечивает такой режим работы, когда информация записывается и считывается по принципу *«последним записан — первым считан»* (LIFO — Last In First Out). Память с подобной организацией широко применяется для запоминания и восстановления содержимого регистров процессора (контекста) при обработке подпрограмм и прерываний. Работу стековой памяти поясняет рис. 5.18, *а*.

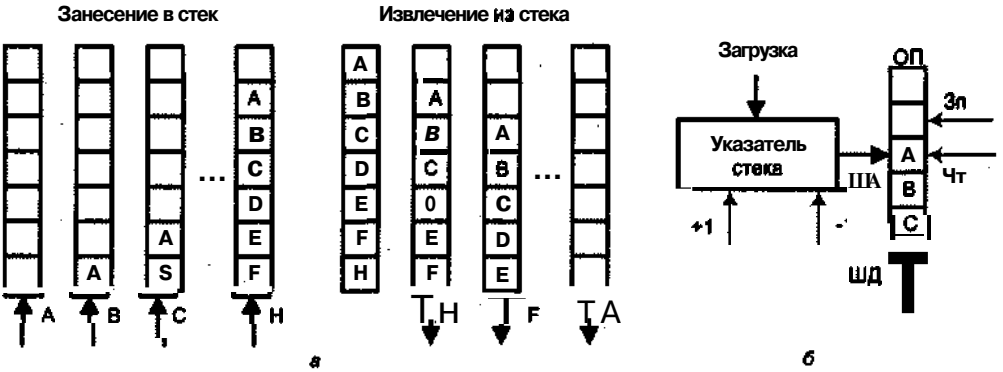


Рис. 5.18. Организация стековой памяти: а - логика работы; *б* - аппаратно-программный стек

Когда слово А заносится в стек, оно располагается в первой свободной ячейке. Каждое следующее записываемое слово перемещает все содержимое стека на одну ячейку вверх и занимает освободившуюся ячейку. Запись очередного кода, после Н, приводит к переполнению стека и потере кода А. Считывание кодов из стека осуществляется в обратном порядке, то есть начиная с кода Н, который был записан последним. Отметим, что доступ к произвольному коду в стеке формально недопустим до извлечения всех кодов, записанных позже.

Наиболее распространенным в настоящее время является внешний или аппаратно-программный стек, в котором для хранения информации отводится область ОП. Обычно для этих целей отводится участок памяти с наибольшими адресами, а стек расширяется в сторону уменьшения адресов. Поскольку программа обычно загружается, начиная с меньших адресов, такой прием во многих случаях позволяет избежать перекрытия областей программы и стека. Адресация стека обеспечивается специальным регистром — *указателем стека* (SP — stack pointer), в который предварительно помещается наибольший адрес области основной памяти, отведенной под стек (рис. 5.18, *б)*.

При занесении в стек очередного слова сначала производится уменьшение на единицу содержимого указателя стека (УС), которое затем используется как адрес ячейки, куда и производится запись, то есть указатель стека хранит адрес той ячейки, к которой было произведено последнее обращение. Это можно описать в виде: УС:= УС - 1; ОП[(УС)]:=ШД.

При считывании слова из стека в качестве адреса этого слова берется текущее содержимое указателя стека, а после того как слово извлечено, содержимое УС

увеличивается на единицу. Таким образом, при извлечении слова из стека реализуются следующие операции: Ш Д:= ОП[(УС)]; УС := УС +1.

Ассоциативная память

В рассмотренных ранее видах запоминающих устройств доступ к информации требовал указания адреса ячейки. Зачастую значительно удобнее искать информацию не по адресу, а опираясь на какой-нибудь характерный признак, содержащийся в самой информации. Такой принцип лежит в основе ЗУ, известного как *ассоциативное запоминающее устройство* (АЗУ). В литературе встречаются и иные названия подобного ЗУ: память, адресуемая по содержанию (content addressable memory); память, адресуемая по данным (data addressable memory); память с параллельным поиском (parallel search memory); каталоговая память (catalog memory); информационное ЗУ (information storage); тегированная память (tag memory). Ассоциативное ЗУ - это устройство, способное хранить информацию, сравнивать ее с некоторым заданным образцом и указывать на их соответствие или несоответствие друг другу. Признак, по которому производится поиск информации, будем называть *ассоциативным признаком,* а кодовую комбинацию, выступающую в роли образца для поиска, - *признаком поиска.* Ассоциативный признак может быть частью искомой информации или дополнительно придаваться ей. В последнем случае его принято называть *тегом* или *ярлыком.*

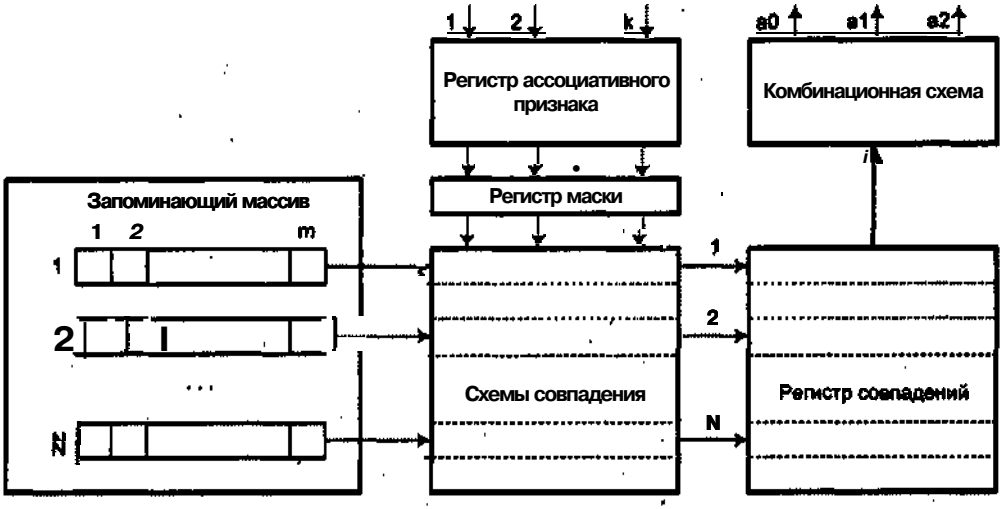


Рис. 5.19. Структура ассоциативного запоминающего устройства

Один из вариантов построения ассоциативной памяти показан на рис. 5.19. АЗУ включает в себя:

- запоминающий массив для хранения N m-разрядных слов, в каждом из которых несколько младших разрядов занимает служебная информация;
- регистр ассоциативного признака, куда помещается код искомой информации (признак поиска). Разрядность регистра k обычно меньше длины слова m;

- схемы совпадения, используемые для параллельного сравнения каждого бита всех хранимых слов с соответствующим битом признака поиска и выработки сигналов совпадения;
- регистр совпадений, где каждой ячейке запоминающего массива соответствует один разряд, в который заносится единица, если все разряды соответствующей ячейки совпали с одноименными разрядами признака поиска;
- регистр маски, позволяющий запретить сравнение определенных битов;
- комбинационную схему, которая на основании анализа содержимого регистра совпадений формирует сигналы, характеризующие результаты поиска информации.

При обращении к АЗУ сначала в регистре маски обнуляются разряды, которые не должны учитываться при поиске информации. Все разряды регистра совпадений устанавливаются в единичное состояние. После этого в регистр ассоциативного признака заносится код искомой информации (признак поиска) и начинается ее поиск, в процессе которого схемы совпадения одновременно сравнивают первый бит всех ячеек запоминающего массива с первым битом признака поиска. Те схемы, которые зафиксировали несовпадение, формируют сигнал, переводящий соответствующий бит регистра совпадений в нулевое состояние. Так же происходит процесс поиска и для остальных незамаскированных битов признака поиска. В итоге единицы сохраняются лишь в тех разрядах регистра совпадений, которые соответствуют ячейкам, где находится искомая информация. Конфигурация единиц в регистре совпадений используется в качестве адресов, по которым производится считывание из запоминающего массива.

Из-за того что результаты поиска могут оказаться неоднозначными, содержимое регистра совпадений подается на комбинационную схему, где формируются сигналы, извещающие о том, что искомая информация:

- *aO — не* найдена;
- *a 1* - содержится в одной ячейке;
- *a2* - содержится более чем в одной ячейке.

Формирование содержимого регистра совпадений и сигналов *aO, al, a2* носит название *операции контроля ассоциации.* Она является составной частью операций считывания и записи, хотя может иметь и самостоятельное значение.

При считывании сначала производится контроль ассоциации по аргументу поиска. Затем, при *aO=1* считывание отменяется из-за отсутствия искомой информации, при *al* = 1 считывается слово, на которое указывает единица в регистре совпадений, а при *a2*= 1 сбрасывается самая старшая единица в регистре совпадений и извлекается соответствующее ей слово. Повторяя эту операцию, можно последовательно считать все слова.

Запись в АП производится без указания конкретного адреса, в первую свободную ячейку. Для отыскания свободной ячейки выполняется операция считывания, в которой не замаскированы только служебные разряды, показывающие, как давно производилось обращение к данной ячейке, и свободной считается либо пустая ячейка, либо та, которая дольше всего не использовалась.

Главное преимущество ассоциативных ЗУ определяется тем, что время поиска информации зависит только от числа разрядов в признаке поиска и скорости опроса разрядов и не зависит от числа ячеек в запоминающем массиве.

Общность идеи ассоциативного поиска информации отнюдь не исключает разнообразия архитектур АЗУ. Конкретная архитектура определяется сочетанием четырех факторов: вида поиска информации; техники сравнения признаков; способа считывания информации при множественных совпадениях и способа записи информации.

В каждом конкретном применении АЗУ задача поиска информации может формулироваться по-разному (рис. 5.20.).

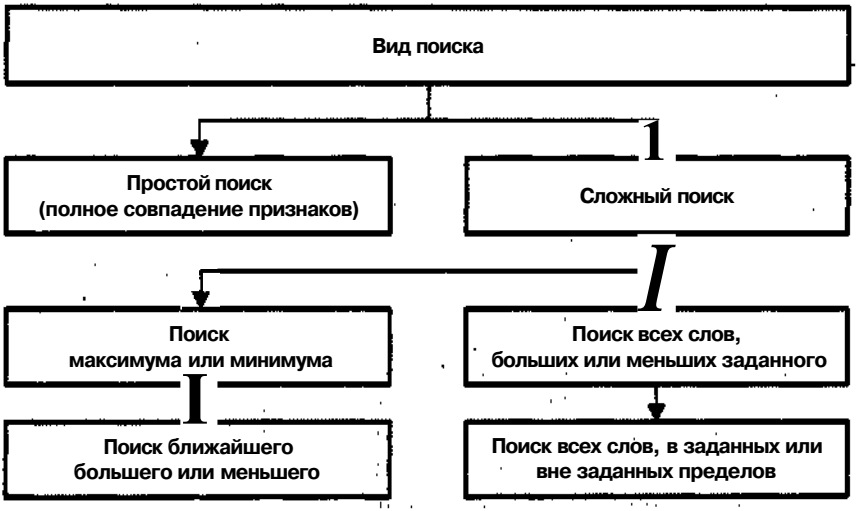


Рис. 5.20. Классификация АЗУ по виду поиска информации в запоминающем массиве

При простом поиске требуется полное совпадение всех разрядов признака поиска с одноименными разрядами слов, хранящихся в запоминающем массиве.

При соответствующей организации АЗУ способно к более сложным вариантам поиска, с частичным совпадением. Можно, например, ставить задачу поиска слов с максимальным или минимальным значением ассоциативного признака. Многократное выборка из АЗУ слова с максимальным или минимальным значением ассоциативного признака (с исключением его из дальнейшего поиска), по существу, представляет собой упорядоченную выборку информации. Упорядоченную выборку можно обеспечить и другим способом, если вести поиск слов, ассоциативный признак которых по отношению к признаку опроса является ближайшим большим или меньшим значением.

Еще одним вариантом сложного поиска может быть нахождение слов, численное значение признака в которых больше или меньше заданной величины. Подобный подход позволяет вести поиск слов с признаками, лежащими внутри или вне заданных пределов..

Очевидно, что реализация сложных методов поиска; связана с соответствующими изменениями в архитектуре АЗУ, в частности с усложнением схемы ЗУ и введением в нее дополнительной логики.

При построении АЗУ выбирают из четырех вариантов организации опроса содержимого памяти (рис. 5.21). Варианты эти могут комбинироваться параллельно по группе разрядов и последовательно по группам. В плане времени поиска наиболее эффективным можно считать параллельный опрос как по словам, так и по разрядам, но не все виды запоминающих элементов допускают такую возможность.

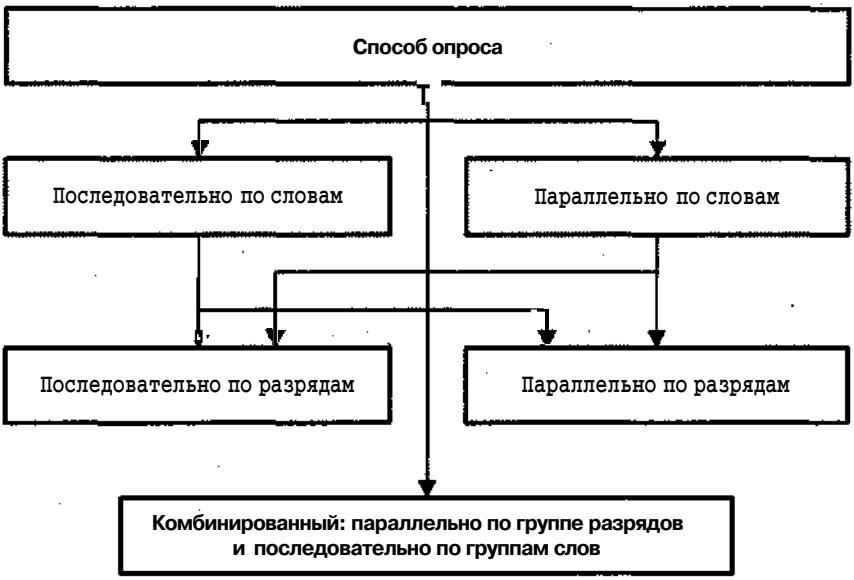


Рис. 5.21. Классификация АЗУ по способу опроса содержимого запоминающего массива

Важным моментом является организация считывания из АЗУ информации, когда с признаком поиска совпадают ассоциативные признаки нескольких слов. В этом случае применяется один из двух подходов (рис. 5.22).

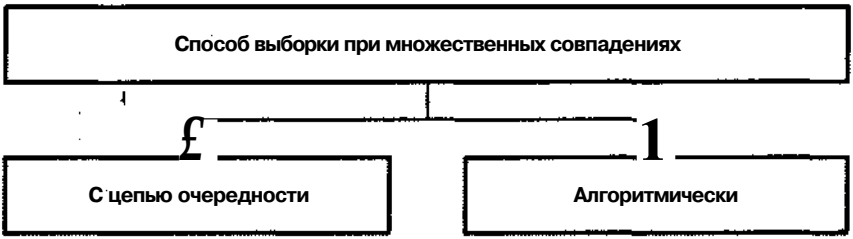


Рис. 5.22. Классификация АЗУ по способу выборки совпавших слов при множественных совпадениях

Цепь очередности реализуется с помощью достаточно сложного устройства, где фиксируются слова, образующие многозначный ответ. Цепь очередности позволяет производить считывание слов в порядке возрастания номера ячейки АЗУ независимо от величины ассоциативных признаков.

При алгоритмическом способе извлечения многозначного ответа выборка производится в результате серии опросов. Серия опросов может формироваться пу-

тем упорядочивания тех разрядов, которые были замаскированы и не участвовали в признаке поиска. В другом варианте для этих целей выделяются специальные разряды. Существует целый ряд алгоритмов, позволяющих организовать упорядоченную выборку из АЗУ. Подробное их описание и сравнительный анализ можно найти в [14].

Существенные отличия в архитектурах АЗУ могут быть связаны с выбранным принципом записи информации. Ранее был описан вариант с записью в незанятую ячейку с наименьшим номером. На практике применяются и иные способы (рис. 5.23), из которых наиболее сложный - запись с сортировкой информации на входе АЗУ по величине ассоциативного признака. Здесь местоположение ячейки, куда будет помещено новое слово, зависит от соотношения ассоциативных признаков вновь записываемого слова и уже хранящихся в АЗУ слов.

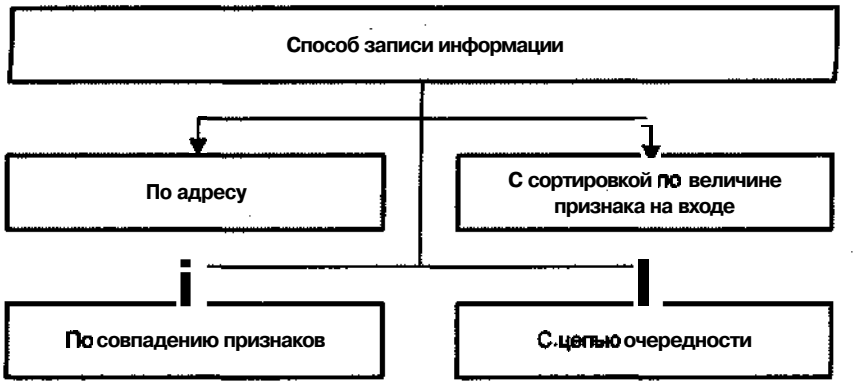


Рис. 5.23. Классификация АЗУ по способу записи информации

Из-за относительно высокой стоимости АЗУ редко используется как самостоятельный вид памяти.

Кэш-память

Как уже отмечалось, в качестве элементной базы основной памяти в большинстве ВМ служат микросхемы динамических ОЗУ, на порядок уступающие по быстродействию центральному процессору. В результате процессор вынужден простаивать несколько тактовых периодов, пока информация из ИМС памяти установится на шине данных ВМ. Если ОП выполнить на быстрых микросхемах статической памяти, стоимость ВМ возрастет весьма существенно. Экономически приемлемое решение этой проблемы было предложено М. Уилксом в 1965 году в процессе разработки ВМ Atlas и заключается оно в использовании двухуровневой памяти, когда между ОП и процессором размещается небольшая, но быстродействующая буферная память. В процессе работы такой системы в буферную память копируются те участки ОП, к которым производится обращение со стороны процессора. В общепринятой терминологии - производится *отображение* участков ОП на буферную память. Выигрыш достигается за счет ранее рассмотренного свойства локальности - если отобразить участок ОП в более быстродействующую буферную память

и переадресовать на нее все обращения в пределах скопированного участка, можно добиться существенного повышения производительности ВМ.

Уилкс называл рассматриваемую буферную память подчиненной (slave memory). Позже распространение получил термин *кэш-память* (от английского слова *cache* - убежище, тайник), поскольку такая память обычно скрыта от программиста в том смысле, что он не может ее адресовать и может даже вообще не знать о ее существовании. Впервые кэш-системы появились в машинах модели 85 семейства IBM 360.

В общем виде использование кэш-памяти поясним следующим образом. Когда ЦП пытается прочитать слово из основной памяти, сначала осуществляется поиск копии этого слова в кэше. Если такая копия существует, обращение к ОП не производится, а в ЦП передается слово, извлеченное из кэш-памяти. Данную ситуацию принято называть успешным обращением или *попаданием* (hit). При отсутствии слова в кэше, то есть при неуспешном обращении — *промахе* (miss),— требуемое слово передается в ЦП из основной памяти, но одновременно из ОП в кэш-память пересылается, блок данных, содержащий это слово.

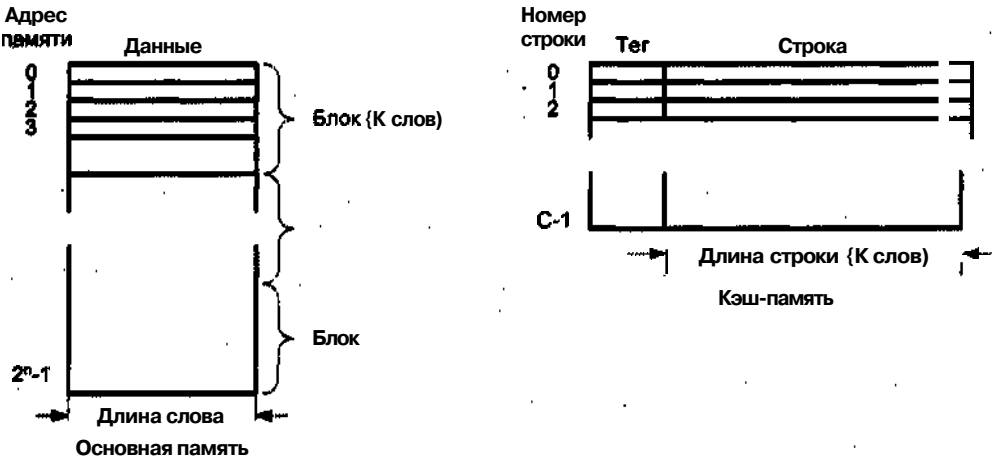


Рис. 5.24. Структура системы с основной и кэш-памятью

На рис. 5.24 приведена структура системы с основной и кэш-памятью. ОП состоит из 2^n адресуемых слов, где каждое слово имеет уникальный *n*-разрядный адрес. При взаимодействии с кэшем эта память рассматривается как *M* блоков фиксированной длины по *K* слов в каждом ($M = 2^n/K$). Кэш-память состоит из С блоков аналогичного размера (блоки в кэш-памяти принято называть *строками),* причем их число значительно меньше числа блоков в основной памяти ($C \ll M$). При считывании слова из какого-либо блока ОП этот блок копируется в одну из строк кэша. Поскольку число блоков ОП больше числа строк, отдельная строка не может быть выделена постоянно одному и тому же блоку ОП. По этой причине каждой строке кэш-памяти соответствует *тег* (признак), содержащий сведения о том, копия какого блока ОП в данный момент хранится в данной строке. В качестве тега обычно используется часть адреса ОП.

На эффективность применения кэш-памяти в иерархической системе памяти влияет целый ряд моментов. К наиболее существенным из них можно отнести:

- емкость кэш-памяти;
- размер строки;
- способ отображения основной памяти на кэш-память;
- алгоритм замещения информации в заполненной кэш-памяти;
- алгоритм согласования содержимого основной и кэш-памяти;
- число уровней кэш-памяти.

Емкость кэш-памяти

Выбор емкости кэш-памяти - это всегда определенный компромисс. С одной стороны, кэш-память должна быть достаточно мала, чтобы ее стоимостные показатели были близки к величине, характерной для ОП. С другой - она должна быть достаточно большой, чтобы среднее время доступа в системе, состоящей из основной и кэш-памяти, определялось временем доступа к кэш-памяти. В пользу уменьшения размера кэш-памяти имеется больше мотивировок. Так, чем вместительнее кэш-память, тем больше логических схем должно участвовать в ее адресации. Как следствие, ИМС кэш-памяти повышенной емкости работают медленнее по сравнению с микросхемами меньшей емкости, даже если они выполнены по одной и той же технологии.

Реальная эффективность использования кэш-памяти зависит от характера решаемых задач, и невозможно заранее определить, какая ее емкость будет действительно оптимальной. Рисунок 5.25, *а* иллюстрирует зависимость вероятности промахов от емкости кэш-памяти для трех программ A, B и C [195]. Несмотря на очевидные различия, просматривается и общая тенденция: по мере увеличения емкости кэш-памяти вероятность промахов сначала существенно снижается, но при достижении определенного значения эффект сглаживается и становится несущественным. Установлено, что для большинства задач близкой к оптимальной является кэш-память емкостью от 1 до 512 Кбайт.

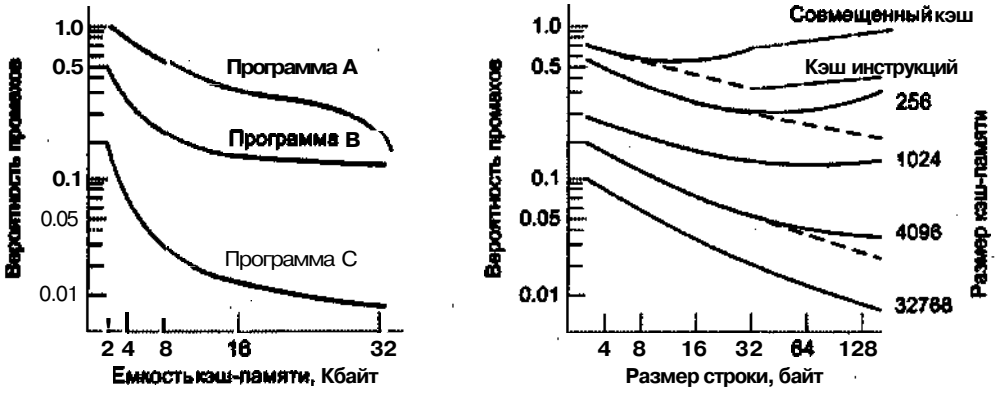


Рис. 5.25. Зависимость вероятности промахов от: а — емкости кэш-памяти; *б* — размера строки

Размер строки

Еще одним важным фактором, влияющим на эффективность использования кэш-памяти, служит размер строки. Когда в кэш-память помещается строка, вместе с требуемым словом туда попадают и соседние слова. По мере увеличения размера строки вероятность промахов сначала падает, так как в кэш, согласно принципу локальности, попадает все больше данных, которые понадобятся в ближайшее время. Однако вероятность промахов начинает расти, когда размер строки становится излишне большим (рис. 5.25, *б)*. Объясняется это тем, что:

- большие размеры строки уменьшают общее количество строк, которые можно загрузить в кэш-память, а малое число строк приводит к необходимости частой их смены;
- по мере увеличения размера строки каждое дополнительное слово оказывается дальше от запрошенного, поэтому такое дополнительное слово менее вероятно понадобится в ближайшем будущем.

Зависимость между размером строки и вероятностью промахов во многом определяется характеристиками конкретной программы, из-за чего трудно рекомендовать определенное значение величины строки. Исследования [183, 195] показывают, что наиболее близким к оптимальному можно признать размер строки, равный 4-8 адресуемым единицам (словам или байтам). На практике размер строки обычно выбирают равным ширине шины данных, связывающей кэш-память с основной памятью.

Способы отображения оперативной памяти на кэш-память

Сущность отображения блока основной памяти на кэш-память состоит в копировании этого блока в какую-то строку кэш-памяти, после чего все обращения к блоку в ОП должны переадресовываться на соответствующую строку кэш-памяти. Удачным может быть признан лишь такой способ отображения, который одновременно отвечает трем требованиям: обеспечивает быструю проверку кэш-памяти на наличие в ней копии блока основной памяти; обеспечивает быстрое преобразование адреса блока ОП в адрес строки кэша; реализует достижение первых двух требований наиболее экономными средствами.

Для облегчения понимания комплекса вопросов, возникающих при выборе способа отображения оперативной памяти на кэш-память, будем рассматривать систему, состоящую из основной памяти емкостью 256 Кслов, и кэш-памяти емкостью 2 Кслова. Для адресации каждого слова основной памяти необходим 18-разрядный адрес (2^{18} =256К). ОП разбивается на блоки по 16 слов в каждом, следовательно, ее удобно рассматривать как линейную последовательность из 16 384 = 2^{14} блоков. При такой организации 18-разрядный адрес можно условно разделить на две части: младшие 4 разряда определяют адрес слова в пределах блока, а старшие 14 — номер одного из 16 384 блоков. Эти старшие 14 разрядов в дальнейшем будем называть *адресом блока основной памяти*. В свою очередь, для адресации любого сло в кэш-памяти требуется 11-разрядный адрес (2^{11}= 2К). Кэш-память разбита на строки такого же размера, что и в ОП (напомним, что применительно к кэш-памяти

вместо слова «блок» принято использовать термин «строка»), то есть содержит $128 = 2^7$ строк. 11-разрядный адрес слова в кэш-памяти также можно представить состоящим из двух частей: адреса слова в строке (4 младших разряда) и *адреса строки кэш-памяти* (7 старших разрядов).

Поскольку ЦП всегда обращается к ОП (кэш-память для ЦП невидима) и формирует для этого 18-разрядный адрес, необходим механизм преобразования такого адреса в 11-разрядный адрес слова в кэше. Так как расположение слов в блоке ОП и строке кэш-памяти идентично, для доступа к конкретному слову в блоке ОП или в строке кэш-памяти можно использовать младшие 4 разряда 18-разрядного адреса. Следовательно, остается лишь задача преобразования 14-разрядного адреса блока основной памяти в 7-разрядный адрес строки кэша.

Известные варианты отображения основной памяти на кэш можно свести к трем видам: прямому, полностью ассоциативному и частично-ассоциативному, причем последний имеет две модификации - множественно-ассоциативное отображение и отображение секторов.

Прямое отображение

При *прямом отображении* адрес строки i кэш-памяти, на которую может быть отображен блоку из ОП, однозначно определяется выражением: $i = j \bmod m$, где m — общее число строк в кэш-памяти. В нашем примере $i = j \bmod 128$, где i может принимать значения от 0 до 127, а адрес блока j — от 0 до 16 383. Иными словами, на строку кэша с номером i отображается каждый 128-й блок ОП, если отсчет начинать с блока, номер которого равен i. Это поясняется рис. 5.26, где основная память условно представлена в виде двухмерного массива блоков, в котором количество рядов равно числу строк в кэш-памяти, а в каждом ряду последовательно перечислены блоки, переадресуемые на одну и ту же строку кэш-памяти.

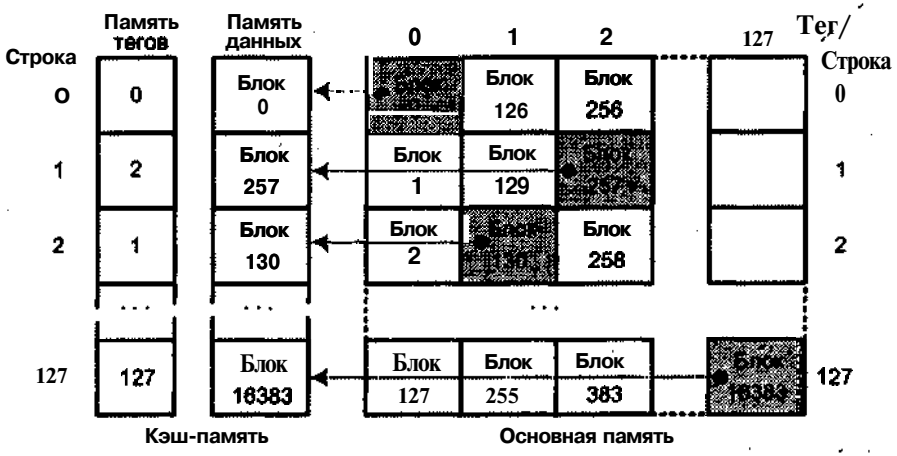


Рис. 5.26. Организация кэш-памяти с прямым отображением

При реализации такого отображения 14-разрядный адрес блока основной памяти условно разбивается на два поля. Логика кэш-памяти интерпретирует эти 14 бит как 7-разрядный тег и 7-разрядное поле строки. Поле строки указывает на

одну из $128 = 2^7$ строк кэш-памяти, а именно на ту, куда может быть отображен блок с заданным адресом. В свою очередь, поле тега определяет, какой именно из списка блоков, закрепленных за данной строкой кэша, будет отображен. Когда блок фактически заносится в память данных кэша, в память тегов кэш-памяти необходимо записать тег этого блока, чтобы отличить его от других блоков, которые могут быть загружены в ту же строку кэша. Тегом служат семь старших разрядов адреса блока.

. Прямое отображение - простой и недорогой в реализации способ отображения. Основной его недостаток — жесткое закрепление за определенными блоками ОП одной строки в кэше. Поэтому если программа поочередно обращается к словам из двух различных блоков, отображаемых на одну и тут же строку кэш-памяти, постоянно станет происходить обновление данной строки и вероятность попадания будет низкой.

Полностью ассоциативное отображение

Полностью ассоциативное отображение позволяет преодолеть недостаток прямо разрешая загрузку любого блока ОП в любую строку кэш-памяти. Логика управления кэш-памяти выделяет в адресе ОП два поля: поле тега и поле слова. Поле тега совпадает с адресом блока основной памяти. Для проверки наличия копии блока в кэш-памяти логика управления кэша должна одновременно проверить теги всех строк на совпадение с полем тега адреса. Этому требованию наилучшим образом отвечает ассоциативная память, то есть тег должен храниться в ассоциативной памяти тегов кэша. Логика работы такой кэш-памяти иллюстрируется рис. 5.27.

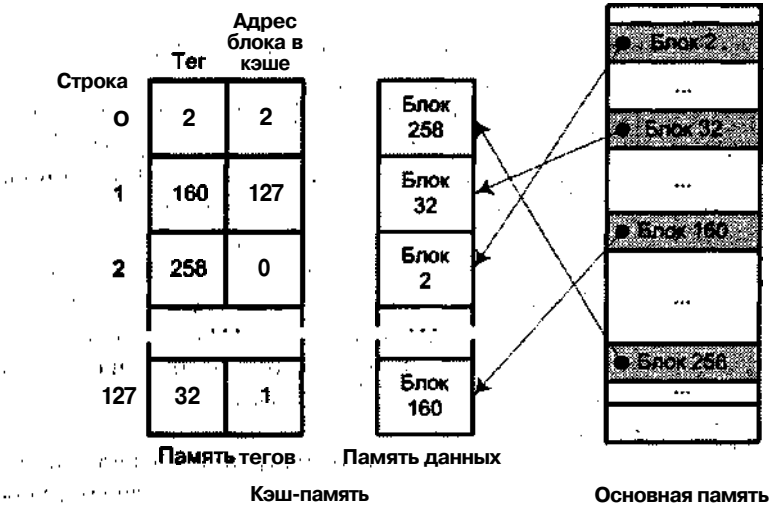


Рис. 5.27. Кэш-память с ассоциативным отображением

Ассоциативное отображение обеспечивает гибкость при выборе строки для вновь записываемого блока. Принципиальный недостаток этого способа - необходимость использования дорогостоящей ассоциативной памяти.

Множественно-ассоциативное отображение

Множественно-ассоциативное отображение относится к группе методов частично-ассоциативного отображения. Оно является одним из возможных компромиссов, сочетающим достоинства прямого и ассоциативного способов отображения и, в известной мере, свободным от их недостатков. Кэш-память (как тегов, так и данных) разбивается на v подмножеств (в дальнейшем будем называть такие подмножества модулями), каждое из которых содержит κ строк (принято говорить, что модуль имеет κ входов). Зависимость между модулем и блоками ОП такая же, как и при прямом отображении: на строки, входящие в модуль i, могут быть отображены только вполне определенные блоки основной памяти, в соответствии с соотношением $i = j \bmod v$, где j — адрес блока ОП. В то же время размещение блоков по строкам модуля — произвольное, и для поиска нужной строки в пределах модуля используется ассоциативный принцип.

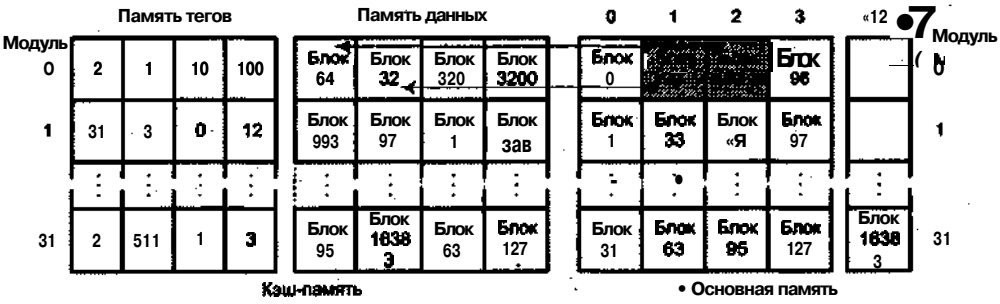


Рис. 5.28. Кэш-память с множественно-ассоциативным отображением

На рис. 5.28 показан пример четырехвходовой кэш-памяти с множественно-ассоциативным отображением. Память данных кэш-памяти разбита на 32 модуля по 4 строки в каждом. Память тегов содержит 32 ячейки, в каждой из которых может храниться 4 значения тегов (по одному на каждую строку модуля). 14-разрядный адрес блока ОП представляется в виде двух полей: 9-разрядного поля тега и 5-разрядного поля номера модуля. Номер модуля однозначно указывает на один из модулей кэш-памяти. Он также позволяет определить номера тех блоков ОП, которые можно отображать на этот модуль. Такими являются блоки ОП, номера которых при делении на 2^5 дают в остатке число, совпадающее с номером данного модуля кэш-памяти! Так, блоки 0,32,64,96 и т. д. основной памяти отображаются на модуль с номером 0; блоки 1, 33, 65, 97 и т. д. отображаются на модуль 1 и т. д. Любой из блоков в последовательности может быть загружен в любую из четырех строк соответствующего модуля.

При такой постановке роль тега выполняют 9 старших разрядов адреса блока ОП, в которых содержится порядковый номер блока в последовательности блоков, отображаемых на один и тот же модуль кэш-памяти. Например, блок 65 в последовательности блоков, отображаемых на модуль 1, имеет порядковый номер 2 (отсчет ведется от 0).

При обращении к кэш-памяти 5-разрядный номер модуля указывает на конкретную ячейку памяти тегов (это соответствует прямому отображению). Далее

производится параллельное сравнение каждого из четырех тегов, хранящихся в этой ячейке, с полем тега поступившего адреса, то есть поиск нужного тега среди четырех возможных осуществляется ассоциативно.

В предельных случаях, когда $v=m,k=1$, множественно-ассоциативное отображение сводится к прямому, а при $v = 1,к = m$ — $к$ ассоциативному.

Наиболее общий вид организации множественно-ассоциативного отображения — использование двух строк на модуль *(v=m/2, k = 2)*. Четырехвходовая множественно-ассоциативная кэш-память ($v = m/4, k = 4$) дает дополнительное улучшение за сравнительно небольшую дополнительную цену [122,164]. Дальнейшее увеличения числа строк в модуле существенного эффекта не привносит.

Следует отметить, что именно этот способ отображения наиболее широко распространен в современных микропроцессорах.

Отображение секторов

Один из первых вариантов частично-ассоциативного отображения, реализованный в модели 85 семейства BM IBM 360. Согласно данному методу, основная память разбивается на секторы, состоящие из фиксированного числа последовательных блоков. Кэш-память также разбивается на секторы, содержащие такое же число строк. Расположение блоков в секторе ОП и секторе кэш-памяти полностью совпадает. Отображение сектора на кэш-память осуществляется ассоциативно, то есть любой сектор из ОП может быть помещен в любой сектор кэш-памяти. Логику работы кэша поясняет рис. 5.29.

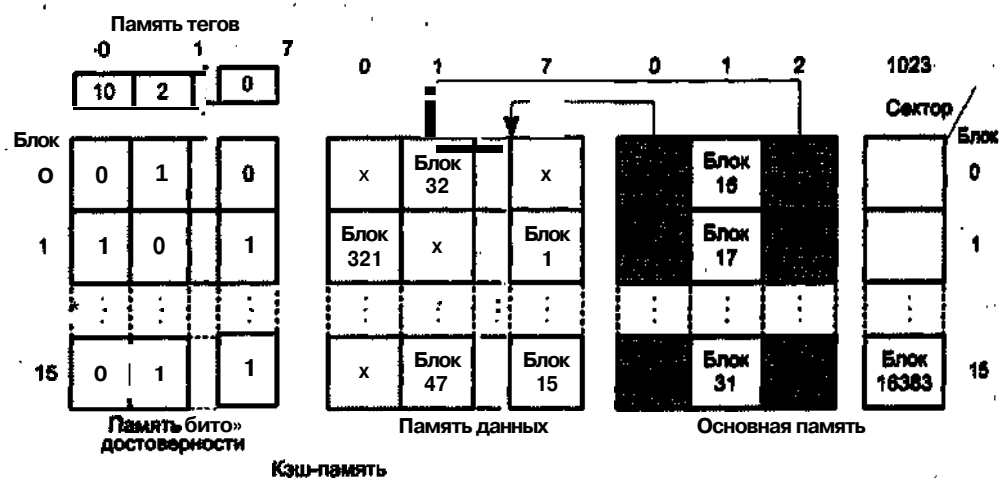


Рис. 5.29. Кэш-память с отображением секторов

Здесь сектор состоит из $16 = 2^4$ блоков по 16 слов, и основная память содержит $1024 = 2^{10}$ сектора. В 14-разрядном адресе блока основной памяти старшие 10 разрядов показывают номер сектора, а младшие 4 — номер блока внутри сектора В свою очередь, кэш-память состоит из $8 = 2^3$ секторов, и 7-разрядный адрес строки в кэше включает в себя адрес сектора кэша (3 старших разряда) и номер блока внутри сектора (4 младших разряда). Ввиду того что расположение блоков в сек-

торах основной и кэш-памяти совпадает, для выбора блока внутри сектора в обоих случаях можно использовать четыре младших разряда адреса блока ОП. В результате задача преобразования адресов сводится к переходу от 10-разрядного номера сектора основной памяти к трехразрядному номеру сектора в кэш-памяти. Такое преобразование осуществляется с помощью ассоциативной памяти тегов на 8 слов (по числу секторов в кэш-памяти). Каждая ячейка памяти тегов содержит 13 разрядов: 10 старших хранят тег — номер сектора ОП, а три младших разряда — номер сектора кэш-памяти, куда отображен данный сектор ОП. Для проверки того, имеется ли копия сектора в кэш-памяти, производится ассоциативный поиск по 10 старшим разрядам памяти тегов. Если произошло совпадение, то для доступа к нужному сектору в памяти данных кэш-памяти используются три младших разряда соответствующей ячейки памяти тегов.

Поскольку размер сектора велик, в рассматриваемой схеме отображения копируется не весь сектор целиком, а только требуемый его блок. Для этого к каждой строке, хранимой в кэш-памяти, добавляется один бит информации, называемый *битом достоверности,* показывающий, относится ли данный блок к текущему сектору или в нем осталась информация из сектора, хранившегося на этом месте до смены секторов.

При заполненной кэш-памяти, когда необходимо заменить один из его секторов, в ассоциативной памяти тегов делается запись о новом секторе, как при замене всего сектора полностью. Фактически же в кэш-память пересылается только тот блок сектора, к которому в данный момент обращается процессор, и в соответствующей строке памяти данных бит достоверности устанавливается в единицу. В то же время биты достоверности остальных строк этого сектора сбрасываются в 0. Далее, по мере копирования других блоков этого сектора, их биты достоверности устанавливаются в единицу.

При такой процедуре заполнения секторов кэш-памяти, в случае обнаружения в ней нужного сектора, предварительно требуется проанализировать состояние бита достоверности соответствующей строки. Если он установлен в 1, значит, обращение к данной ячейке кэш-памяти возможно. При нулевом значении бита достоверности сначала производится копирование нужного блока в кэш-память, его бит достоверности устанавливается в единицу, и лишь после этого разрешается доступ к указанному адресу в кэш-памяти. •

В том случае, когда при замене сектора в кэш-памяти необходимо сохранить его старое содержимое в основной памяти, в ОП пересылаются только заменяемые блоки сектора.

Алгоритмы замещения информации в заполненной кэш-памяти

Когда кэш-память заполнена, занесение в нее нового блока связано с замещением содержимого одной из строк. При прямом отображении каждому блоку основной памяти соответствует только одна определенная строка в кэш-памяти, и никакой иной выбор удаляемой строки здесь невозможен. При полностью и частично ассоциативных способах отображения требуется какой-либо алгоритм замещения (выбора удаляемой из кэш-памяти строки).

Основная цель стратегии замещения — удерживать в кэш-памяти строки, к которым наиболее вероятны обращения в ближайшем будущем, и заменять строки, доступ к которым произойдет в более отдаленном времени или вообще не случится. Очевидно, что оптимальным будет алгоритм, который замещает ту строку, обращение к которой в будущем произойдет позже, чем к любой другой строке кэша К сожалению, такое предсказание практически нереализуемо, и приходится привлекать алгоритмы, уступающие оптимальному. Вне зависимости от используемого алгоритма замещения для достижения высокой скорости он должен быть реализован аппаратными средствами.

Среди множества возможных алгоритмов замещения наиболее распространенными являются четыре, рассматриваемые в порядке уменьшения их относительной эффективности.

Наиболее эффективным является алгоритм замещения на основе *наиболее давнего использования* (LRU - Least Recently Used), при котором замещается та строка кэш-памяти, к которой дольше всего не было обращения. Проводившиеся исследования показали, что алгоритм LRU, который «смотрит» назад, работает достаточно хорошо в сравнении с оптимальным алгоритмом, «смотрящим» вперед.

Наиболее известны два способа аппаратурной реализации этого алгоритма.

В первом из них с каждой строкой кэш-памяти ассоциируют счетчик. К содержимому всех счетчиков через определенные интервалы времени добавляется единица. При обращении к строке ее счетчик обнуляется. Таким образом, наибольшее число будет в счетчике той строки, к которой дольше всего не было обращений, и эта строка - первый кандидат на замещение.

Второй способ реализуется с помощью очереди, куда в порядке заполнения строк кэш-памяти заносятся ссылки на эти строки. При каждом обращении к строке ссылка на нее перемещается в конец очереди. В итоге первой в очереди каждый раз оказывается ссылка на строку, к которой дольше всего не было обращений. Именно эта строка прежде всего и заменяется.

Другой возможный алгоритм замещения - алгоритм, работающий по принципу *«первый вошел, первый вышел »* (FIFO - First In First Out). Здесь заменяется строка, дольше всего находившаяся в кэш-памяти. Алгоритм легко реализуется с помощью рассмотренной ранее очереди, с той лишь разницей, что после обращения к строке положение соответствующей ссылки в очереди не меняется.

Еще один алгоритм - замена *наименее часто использовавшейся* строки (LFU - Least Frequently Used). Заменяется та строка в кэш-памяти, к которой было меньше всего обращений. Принцип можно воплотить на практике, связав каждую строку со счетчиком обращений, к содержимому которого после каждого обращения добавляется единица. Главным претендентом на замещение является строка, счетчик которой содержит наименьшее число.

Простейший алгоритм - *произвольный выбор* строки для замены. Замещаемая строка выбирается случайным образом. Реализовано это может быть, например, с помощью счетчика, содержимое которого увеличивается на единицу с каждым тактовым импульсом, вне зависимости от того, имело место попадание или промах. Значение в счетчике определяет заменяемую строку в полностью ассоциативной кэш-памяти или строку в пределах модуля для множественно-ассоциативной кэш-памяти. Данный алгоритм используется крайне редко.

Среди известных в настоящее время систем с кэш-памятью наиболее встречаемым является алгоритм LRU.

Алгоритмы согласования содержимого кэш-памяти и основной памяти

В процессе вычислений ЦП может не только считывать имеющуюся информацию, но и записывать новую, обновляя тем самым содержимое кэш-памяти. С другой стороны, многие устройства ввода/вывода умеют напрямую обмениваться информацией с основной памятью. В обоих вариантах возникает ситуация, когда содержимое строки кэша и соответствующего блока ОП перестает совпадать. В результате на связанное с основной памятью устройство вывода может быть выдана «устаревшая» информация, поскольку все изменения в ней, сделанные процессором, фиксируются только в кэш-памяти, а процессор будет использовать старое содержимое кэш-памяти вместо новых данных, загруженных в ОП из устройства ввода.

Для разрешения первой из рассмотренных ситуаций (когда процессор выполняет операцию записи) в системах с кэш-памятью предусмотрены методы обновления основной памяти, которые можно разбить на две большие группы: *метод сквозной записи* (write through) и *метод обратной записи* (write back).

По методу сквозной записи прежде всего обновляется слово, хранящееся в основной памяти. Если в кэш-памяти существует копия этого слова, то она также обновляется. Если же в кэш-памяти отсутствует нужная копия, то либо из основной памяти в кэш-память пересылается блок, содержащий обновленное слово *(сквозная запись с отображением); либо этого* не делается *(сквозная запись без отображения)*.

Главное достоинство метода сквозной записи состоит в том, что когда строка в кэш-памяти назначается для хранения другого блока, то удаляемый блок можно не возвращать в основную память, поскольку его копия там уже имеется. Метод достаточно прост в реализации. К сожалению, эффект от использования кэш-памяти (сокращение времени доступа) в отношении к операциям записи здесь отсутствует. Данный метод применен в микропроцессорах i486 фирмы Intel.

- Определенный выигрыш дает его модификация, известная *как метод буферизированной сквозной записи*. Информация сначала записывается в кэш-память и в специальный буфер, работающий по схеме FIFO. Запись в основную память производится уже из буфера, а процессор, не дожидаясь ее окончания, может сразу же продолжать свою работу. Конечно, соответствующая логика управления должна заботиться о том, чтобы своевременно «опустошать» заполненный буфер. При использовании буферизации процессор полностью освобождается от работы с ОП.

Согласно методу обратной записи, слово заносится только в кэш-память. Если соответствующей строки в кэш-памяти нет, то нужный блок сначала пересылается из ОП, после чего запись все равно выполняется исключительно в кэш-память. При замещении строки ее необходимо предварительно переслать в соответствующее место основной памяти. Для метода обратной записи, в отличие от алгоритма сквозной записи, характерно то, что при каждом чтении из основной памяти осуществляются две пересылки между основной и кэш-памятью.

У рассматриваемого метода есть разновидность — *метод флаговой обратной записи*. Когда в какой-то строке кэша производится изменение, устанавливается связанный с этой строкой бит изменения (флажок). При замещении строка из кэш-памяти переписывается в ОП только тогда, когда ее флажок установлен в 1. Ясно, что эффективность флаговой обратной записи несколько выше. Такой метод используется в микропроцессорах класса i486 и Pentium фирмы Cyrix.

В среднем обратная запись на 10% эффективнее сквозной записи, но для ее реализации требуются и повышенные аппаратные затраты. С другой стороны, практика показывает, что операции записи составляют небольшую долю от общего количества обращений к памяти. Так, в [194] приводится число 16%. Другие авторы оценивают долю операций записи величинами в диапазоне от 5 до 34%. Таким образом, различие по быстродействию между рассмотренными методами невелико.

Теперь рассмотрим ситуацию, когда в основную память из устройства ввода, минуя процессор, заносится новая информация и неверной становится копия, хранящаяся в кэш-памяти. Предотвратить подобную несогласованность позволяют два приема. В первом случае система строится так, чтобы ввод любой информации в ОП автоматически сопровождался соответствующими изменениями в кэш-памяти. Для второго подхода «прямой» доступ к основной памяти допускается только через кэш-память.

Смешанная и разделенная кэш-память

Когда в микропроцессорах впервые стали применять внутреннюю кэш-память, ее обычно использовали как для команд, так и для данных; Такую кэш-память Принято называть *смешанной,* а соответствующую архитектуру - *Принстонской* (Princeton architecture), по названию университета, где разрабатывались ВМ с единой памятью для команд и данных, то есть соответствующие классической архитектуре фон-Неймана. Сравнительно недавно стало обычным разделять кэш-память на две — отдельно для команд и отдельно для данных. Подобная архитектура получила название *Гарвардской* (Harvard architecture), поскольку именно в Гарвардском университете был создан компьютер «Марк-1» (1950 год), имевший раздельные ЗУ для команд и данных.

Смешанная кэш-память обладает тем преимуществом, что при заданной емкости ей свойственна более высокая вероятность попаданий по сравнению с разделенной, поскольку в ней оптимальный баланс между командами и данными устанавливается автоматически. Так, если в выполняемом фрагменте программы обращения к памяти связаны в основном с выборкой команд, а доля обращений к данным относительно мала, кэш-память имеет тенденцию насыщаться командами, и наоборот.

С другой стороны, при раздельной кэш-памяти выборка команд и данных может производиться одновременно, при этом исключаются возможные конфликты. Последнее обстоятельство существенно в "системах, использующих' конвейеризацию команд, где процессор извлекает команды с опережением и заполняет ими буфер или конвейер.

В табл. 5.7 приведены основные параметры внутренней кэш-памяти для наиболее распространенных типов микропроцессоров.

Микропроцессор	Размер, Кбайт
DEC Alpha 21064	8 - К / 8 - Д
IBM PowerPC 601	32-С
IBM PowerPC 603	8 - К / 8 - Д
IBM PowerPC 604	16 - К / 16 - Д
IBM PowerPC 620	32 - К/32-Д
Intel 486DX2	8 - С
Intel Pentium	8 - К / 8 - Д
MIPS R4000SC	8 - К / 8 - Д
MIPS R10000	32 - К / 32 - Д
Motorola 88110	8 - К / 8 - Д
SUNSuperSPARC	20 - К / 16 - Д
SUN UltraSPARC	16 - К / 16 - Д

Таблица 5.7. Организация кэ

Следует добавить, что в некоторых ВМ, помимо кэш-памяти команд и кэш-памяти данных, может использоваться и адресная кэш-память (в устройствах управления памятью и при преобразовании виртуальных адресов в физические).

Одноуровневая и многоуровневая кэш-память

Современные технологии позволяют разместить кэш-память и ЦП на общем кристалле. Такая внутренняя кэш-память строится по технологии статического ОЗУ и является наиболее быстродействующей. Емкость ее обычно не превышает 64 Кбайт. Попытки увеличения емкости обычно приводят к снижению быстродействия, главным образом из-за усложнения схем управления и дешифрации адреса. Общую емкость кэш-памяти ВМ увеличивают за счет второй (внешней) кэш-памяти, расположенной между внутренней кэш-памятью и ОП. Такая система известна под названием двухуровневой, где внутренней кэш-памяти отводится роль первого уровня (L1), а внешней - второго уровня (L2). Емкость L2 обычно на порядок больше, чем у L1, а быстродействие и стоимость - несколько ниже. Память второго уровня также строится как статическое ОЗУ. Типичная емкость кэш-памяти второго уровня — 256 и 512 Кбайт, реже — 1 Мбайт, а реализуется она, как правило, в виде отдельной микросхемы, хотя в последнее время L2 часто размещают на одном кристалле с процессором, за счет чего сокращается длина связей и повышается быстродействие.

При доступе к памяти ЦП сначала обращается к кэш-памяти первого уровня. В случае промаха производится обращение к кэш-памяти второго уровня. Если информация отсутствует и в L2, выполняется обращение к ОП и соответствующий блок заносится сначала в L2, а затем и в L1. Благодаря такой процедуре часто запрашиваемая информация может быть быстро восстановлена из кэш-памяти второго уровня.

1К— кэш-память команд, Д — кэш-память данных, С — смешанная кэш-память.

Среднее время доступа $T_{ср}$ к одноуровневой кэш-памяти можно оценить как:

$$T_{ср} = T_{L1h} + K_{L1m} T_{L1m},$$

где T_{L1h} - время обращения при попадании; K_{L1m}—коэффициент промахов; T_{L1m} - потери на промах. Для двухуровневой кэш-памяти имеем:

$$T_{ср} = T_{L1h} + (K_{L1m} T_{L2h}) + (K_{L2m} T_{L2m}).$$

Потенциальная экономия за счет применения L2 зависит от вероятности попаданий как в L1, так и в L2. Ряд исследований показывает, что использование кэш-памяти второго уровня существенно улучшает производительность.

В большинстве семейств микропроцессоров предусмотрены специальные ИМС контроллеров внешней кэш-памяти, например микросхема 8291 для микропроцессора Intel 486 или 82491 - для Intel Pentium. Для ускорения обмена информацией между ЦП и L2 между ними часто вводят специальную шину, так называемую *шину заднего плана,* в отличие от *шины переднего плана,* связывающей ЦП с основной памятью.

Количество уровней кэш-памяти не ограничивается двумя. В некоторых ВМ уже можно встретить кэш-память третьего уровня (L3) и ведутся активные дискуссии о введении также и кэш-памяти четвертого уровня (L4). Характер взаимодействия очередного уровня с предшествующим аналогичен описанному для L1 и L2. Таким образом, можно говорить об иерархии кэш-памяти. Каждый последующий уровень характеризуется большей емкостью, меньшей стоимостью, но и меньшим быстродействием, хотя оно все же выше, чем у ЗУ основной памяти.

Дисковая кэш-память

Концепция кэш-памяти применима и к дисковым ЗУ. Принцип кэширования дисков во многом схож с принципом кэширования основной памяти, хотя способы доступа к диску и ОП существенно разнятся. Если время обращения к любой ячейке ОП одинаково, то для диска оно зависит от целого ряда факторов. Во-первых, нужно затратить некоторое время для установки головки считывания/записи на нужную дорожку. Во-вторых, поскольку при движении головка вибрирует, необходимо подождать, чтобы она успокоилась. В-третьих, искомый сектор может оказаться под головкой также лишь спустя некоторое время.

Дисковая кэш-память представляет собой память с произвольным доступом, «размещенную» между дисками и ОП. Емкость такой памяти обычно достаточно велика - от 8 Мбайт и более. Пересылка информации между дисками и основной памятью организуется контроллером дисковой кэш-памяти. Изготавливается дисковая кэш-память на базе таких же полупроводниковых запоминающих устройств, что и основная память, поэтому в ряде случаев с ней обращаются как с дополнительной основной памятью. С другой стороны, в ряде операционных систем, таких как UNIX, в качестве дискового кэша используется область основной памяти.

В дисковой кэш-памяти хранятся блоки информации, которые с большой вероятностью будут востребованы в ближайшем будущем. Принцип локальности, обеспечивающий эффективность обычной кэш-памяти, справедлив и для дисковой, приводя к сокращению времени ввода/вывода данных от величин 20-30 мс до значений порядка 2-5 мс, в зависимости объема передаваемой информации.

В качестве единицы пересылки может выступать сектор, несколько секторов, а также одна или несколько дорожек диска. Кроме того, иногда применяется пересылка информации, начиная с выбранного сектора на дорожке до ее конца. В случае пересылки секторов кэш-память заполняется не только требуемым сектором, но секторами, непосредственно следующими за ним, так как известно, что в большинстве случаев взаимосвязанные данные хранятся в соседних секторах. Этот метод известен также как *опережающее чтение* (read ahead).

В дисковых кэшах обычно используется алгоритм сквозной записи. Специфика состоит в том, что далеко не всю информацию, перемещаемую между дисками и основной памятью, выгодно помещать в дисковый кэш. В ряде случаев определенные данные и команды целесообразно пересылать напрямую между ОП и диском По этой причине в системах с дисковым кэшем предусматривают специальный динамический механизм, позволяющий переключать тракт пересылки информации: через кэш или минуя его.

Одна из привлекательных сторон дискового кэша в том, что связанные с ним преимущества могут быть получены без изменений в имеющемся аппаратном и программном обеспечении. Многие серийно выпускаемые дисковые кэши интегрированы в состав дисковых ЗУ. В качестве примера можно привести модель 23 системы IBM 3880, в состав которой входят дисковые ЗУ со встроенным контроллером кэш-памяти и кэш-памятью емкостью от 8 до 64 Мбайт. Дисковая кэш-память применяется и в персональных ВМ.

Дисковая кэш-память обычно включает в себя средства для обнаружения и исправления ошибок. Так, в уже упоминавшейся модели 23 системы IBM 3880 имеются средства для обнаружения тройных ошибок (одновременного появления ошибок в трех разрядах) и исправления одиночных, двойных и большинства тройных ошибок. Более ранняя модель 13 той же системы имела кэш-память емкостью 4-8 Мбайт и умела обнаруживать двойные и исправлять одиночные ошибки. В обеих моделях замещение информации в кэше производится в соответствии с алгоритмом минимального предыдущего использования (LRU).

Примечательно, что архитектура кэш-памяти современных магнитных дисков типа «винчестер» реализует полностью ассоциативное отображение.

Понятие виртуальной памяти

Для большинства типичных применений ВМ характерна ситуация, когда размещение всей программы в ОП невозможно из-за ее большого размера. В этом, однако, и нет принципиальной необходимости, поскольку в каждый момент времени «внимание» машины концентрируется на определенных сравнительно небольших участках программы. Таким образом, в ОП достаточно хранить только используемые в данный период части программ, а остальные части могут располагаться на внешних ЗУ(ВЗУ). Сложность подобного подхода в том, что процессы обращения к ОП и ВЗУ существенно различаются, и это усложняет задачу программиста. Выходом из такой ситуации было появление в 1959 году идеи *виртуализации памяти* [88], под которой понимается метод автоматического управления иерархической памятью, при котором программисту кажется, что он имеет дело с единой

памятью большой емкости и высокого быстродействия. Эту память называют виртуальной (кажущейся) памятью. По своей сути виртуализация памяти представляет собой способ аппаратной и программной реализации концепции иерархической памяти.

В рамках идеи виртуализации памяти ОП рассматривается как линейное пространство N адресов, называемое *физическим пространством* памяти. Для задач, где требуется более чем *N* ячеек, предоставляется значительно большее пространство адресов (обычно равное общей емкости всех видов памяти), называемое *виртуальным пространством,* в общем случае не обязательно линейное. Адреса виртуального пространства называют *виртуальными,* а адреса физического пространства - *физическими.* Программа пишется в виртуальных адресах, но поскольку для её выполнения нужно, чтобы обрабатываемые команды и данные находились в ОП, требуется, чтобы каждому виртуальному адресу соответствовал физический. Таким образом, в процессе вычислений необходимо, прежде всего, переписать из ВЗУ в ОП ту часть информации, на которую указывает виртуальный адрес (отобразить виртуальное пространство на физическое), после чего преобразовать виртуальный адрес в физический (рис. 5.30).

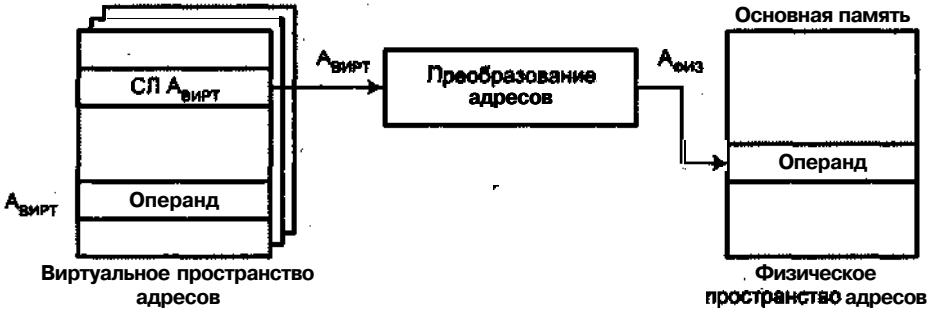


Рис. 5.30. Отображение виртуального адреса на физический

Среди систем виртуальной памяти можно выделить два класса: системы с фиксированным размером блоков (страничная организация) и системы с переменным размером блоков (сегментная организация). Оба варианта обычно совмещают (сегментно-страничная организация).

Страничная организация памяти

Целям преобразования виртуальных адресов в физические служит страничная организация памяти. Ее идея состоит в разбиении программы на части равной величины, называемые *страницами.* Размер страницы обычно выбирают в пределах 4- 8 Кбайт, но так, чтобы он был кратен емкости одного сектора магнитного диска. Виртуальное и физическое адресные пространства разбиваются на блоки размером в страницу. Блок основной памяти, соответствующий странице, часто называют *страничным кадром* или *фреймом* (page frame). Страницам виртуальной и физической памяти присваивают номера. Процесс доступа к данным по их виртуальному адресу иллюстрирует рис . 5.31.

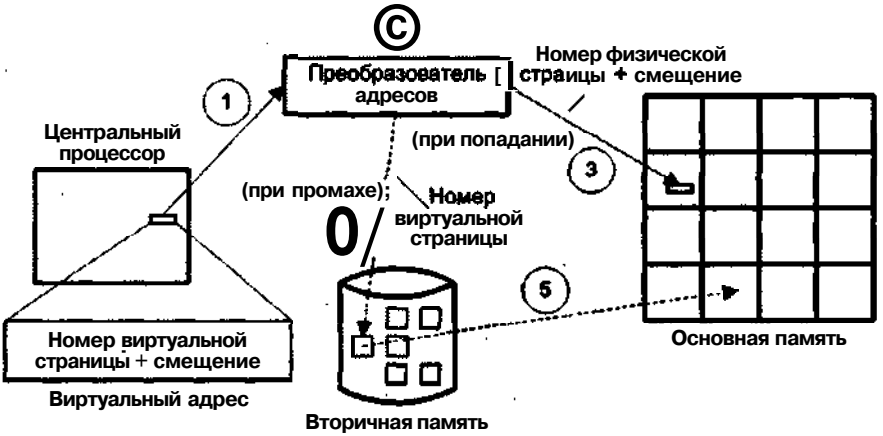


Рис. 5 . 3 1 . Страничная организация виртуальной памяти

Центральный процессор обращается к ячейке, указав ее виртуальный адрес ф, состоящий из номера виртуальной страницы и смещения относительно ее начала. Этот адрес поступает в систему преобразования адресов 2, с целью получения из него физического адреса ячейки в основной памяти 3. Поскольку смещение в виртуальном и физическом адресе одинаковое, преобразованию подвергается лишь номер страницы. Если преобразователь обнаруживает, что нужная физическая страница отсутствует в основной памяти (произошел промах или страничный сбой), то нужная страница считывается из внешней памяти и заносится в ОП (4, 5).

Преобразователь адресов - это часть операционной системы, транслирующая номер виртуальной страницы в номер физической страницы, расположенной в основной памяти, а также аппаратура, обеспечивающая этот процесс и позволяющая ускорить его. Преобразование осуществляется с помощью так называемой *страничной таблицы.* При отсутствии нужной страницы в ОП преобразователь адресов вырабатывает признак страничного сбоя, по которому операционная система приостанавливает вычисления, пока нужная страница не будет считана из вторичной памяти и помещена в основную.

Виртуальное пространство полностью описывается двумя таблицами (рис. 5.32): страничной таблицей и картой диска (будем считать, что вторичная память реализована на магнитных дисках). Таблица страниц определяет, какие виртуальные страницы находятся в основной памяти и в каких физических фреймах, а карта диска содержит информацию о секторах диска, где хранятся виртуальные страницы на диске.

Число записей в страничной таблице (СТ) в общем случае равно количеству виртуальных страниц. Каждая запись содержит поле номера физической страницы (НФС) и четыре признака: V, R, M и A.

Признак присутствия V устанавливается в единицу, если виртуальная страница в данный момент находится в основной памяти. В этом случае в поле номера физической страницы находится соответствующий номер. Если V = 0, то при попытке обратиться к данной виртуальной странице преобразователь адреса генери-

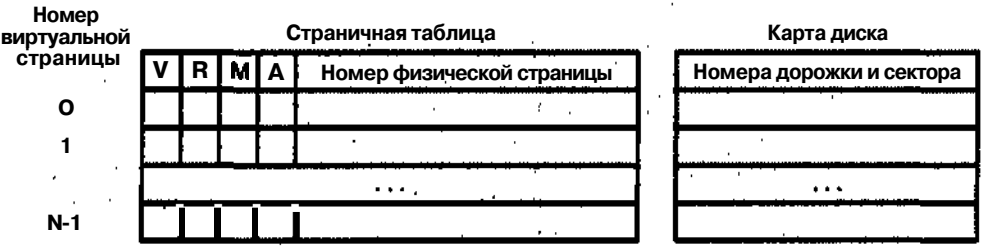


Рис. 5.32. Структура страничной таблицы и карты диска

рует сигнал страничного сбоя (page fault), и операционная система предпринимает действия по загрузке страницы с диска в ОП, обращаясь для этого к карте диска. В карте указано, на какой дорожке и в каком секторе диска расположена каждая из виртуальных страниц[1]. Загрузка страницы с диска в ОП сопровождается записью в соответствующую строку страничной таблицы (указывается номер физической страницы, куда была загружена виртуальная страница).

В принципе карта диска может быть совмещена со страничной таблицей путем добавления к последней еще одного поля. Другим вариантом может быть увеличение разрядности поля номера физической страницы и хранение в нем номеров дорожек и секторов для виртуальных страниц, отсутствующих в основной памяти. В этом случае вид хранимой информации будет определять признак V.

Признак использования страницы R устанавливается при обращении к данной станице. Эта информация используется в алгоритме замещения страниц для выбора той из них, которую можно наиболее безболезненно удалить из ОП, чтобы освободить место для новой. Проблемы замещения информации в ОП решаются так же, как и для кэш-памяти.

Поскольку в ОП находятся лишь копии страниц, а их оригиналы хранятся на диске, необходимо обеспечить идентичность подлинников и копий. В ходе вычислений содержимое отдельных страниц может изменяться, что фиксируется путем установки в единицу *признака модификации* M. При удалении страницы из ОП проверяется состояние признака M. Если M = 1, то перед удалением страницы из основной памяти ее необходимо переписать на диск, а при M =О этого можно не делать.

Признак прав доступа A служит целям защиты информации и определяет, какой вид доступа к странице разрешен: только для чтения, только для записи, для чтения и для записи.

Когда программа загружается в ОП, она может быть направлена в любые свободные в данный момент страничные кадры, независимо от того, расположены ли они подряд или нет. Страничная организация позволяет сократить объем пересылок информации между внешней памятью и ОП, так как страницу не нужно загружать до тех пор, пока она действительно не понадобится.

Способ реализации СТ жизненно важен для эффективности техники виртуальной адресации, поскольку каждое обращение к памяти предполагает обраще-

[1] Множество страниц присутствующих в данный момент в основной памяти, является подмножеством всех виртуальных страниц, хранящихся на магнитном диске.

ние к страничной таблице. Наиболее быстрый способ - хранение таблицы в специально выделенных для этого регистрах, но от него приходится отказываться при большом объеме СТ. Остается практически один вариант — выделение страничной таблице области основной памяти, несмотря на то что это приводит к двухкратному увеличению времени доступа к информации. Чтобы сократить это время, в состав ВМ включают дополнительное ЗУ, называемое *буфером быстрого преобразования адреса* (TLB - Translation Look-aside Buffer), или *буфером ассоциативной трансляции,* или *буфером опережающей выборки* и представляющее собой кэш-память. При каждом преобразовании номера виртуальной страницы в номер физической страницы результат заносится в TLB: номер физической страницы в память данных, а виртуальной - в память тегов. Таким образом, в TLB попадают результаты нескольких последних операций трансляции адресов. При каждом обращении к ОП преобразователь адресов сначала производит поиск в памяти тегов TLB номера требуемой виртуальной страницы. При попадании адрес соответствующей физической страницы берется из памяти данных TLB. Если в TLB зафиксирован промах, то процедура преобразования адресов производится с помощью страничной таблицы, после чего осуществляется запись новой пары «номер виртуальной страницы - номер физический страницы» в TLB. Структура TLB показана на рис. 5.33.

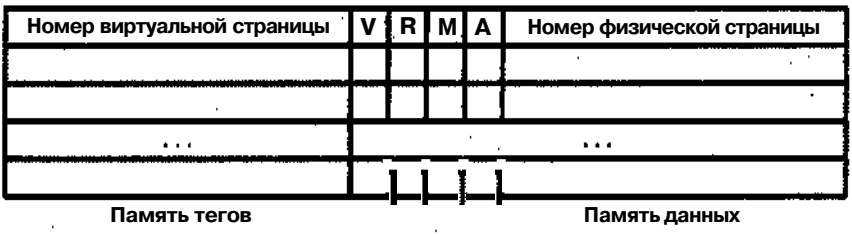


Рис. 5.33. Структура буфера быстрого преобразования адресов

Буфер преобразования адресов обычно реализуется в виде полностью ассоциативной или множественно-ассоциативной кэш-памяти с высокой степенью ассоциативности и временем доступа, сопоставимым с аналогичным показателем для кэш-памяти первого уровня (L1). Число входов у типовых TLB невелико (64-256). Так, TLB микропроцессора Pentium III имеет 64 входа при размере страницы 4 Кбайт, что позволяет получить быстрый доступ к адресному пространству в 256 Кбайт.

Серьезной проблемой в системе виртуальной памяти является большой объем страничных таблиц, который пропорционален числу виртуальных страниц. Таблица занимает значительную часть ОП, а на поиск уходит много времени, что крайне нежелательно.

Один из способов сокращения длины таблиц основан на введении многоуровневой организации таблиц. В этом варианте информация оформляется в виде нескольких страничных таблиц сравнительно небольшого объема, которые образуют второй уровень. Первый уровень представлен таблицей с каталогом, где указано местоположение каждой из страничных таблиц (адрес начала таблицы в памяти) второго уровня. Сначала в каталоге определяется расположение нужной странич-

ной таблицы и лишь затем производится обращение к нужной таблице. О достигаемом эффекте можно судить из следующего примера. Пусть адресная шина ВМ имеет ширину 32 бита, а размер страницы равен 4 Кбайт. Тогда количество виртуальных страниц, а следовательно, и число входов в единой страничной таблице составит 2^{20}. При двухуровневой организации можно обойтись одной страницей первого уровня на 1024 (2^{10}) входов и 1024 страничными таблицами на такое же число входов.

Другой подход называют *способом обращенных или инвертированных страничных таблиц*. Такую таблицу в каком-то смысле можно рассматривать как увеличенный эквивалент TLB, отличающийся тем, что она хранится в ОП и реализуется не аппаратурой, а программными средствами. Число входов в таблицу определяется емкостью ОП и равно -числу страниц, которое может быть размещено в основной памяти. Одновременно с этим имеется и традиционная СТ, но хранится она не в ОП, а на диске. Для поиска нужной записи в инвертированной таблице используется хэширование, когда номер записи в таблице вычисляется в соответствии с некой хэш-функцией. Аргументом этой функции служит номер искомой виртуальной страницы. Хэширование позволяет ускорить операцию поиска. Если нужная страница в ОП отсутствует, производится обращение к основной таблице на диске и после загрузки страницы в ОП корректируется и инвертированная таблица.

Сегментно-страничная организация памяти

При страничной организации предполагается, что виртуальная память — это непрерывный массив со сквозной нумерацией слов, что не всегда можно признать оптимальным. Обычно программа состоит из нескольких частей — кодовой, информационной и стековой. Так как заранее неизвестны длины этих составляющих, то удобно, чтобы при программировании каждая из них имела собственную нумерацию слов, отсчитываемых с нуля. Для этого организуют систему *сегментированной памяти,* выделяя в виртуальном пространстве независимые линейные пространства переменной длины, называемые *сегментами*. Каждый сегмент представляет собой отдельную логическую единицу информации, содержащую совокупность данных или программный код и расположенную в адресном пространстве пользователя. В каждом сегменте устанавливается своя собственная нумерация слов, начиная с нуля. Виртуальная память также разбивается на сегменты, с независимой адресацией слов внутри сегмента. Каждой составляющей программы выделяется сегмент памяти. Виртуальный адрес определяется номером сегмента и адресом внутри сегмента. Для преобразования виртуального адреса в физический используется специальная *сегментная таблица*.

Недостатком такого подхода является то, что неодинаковый размер сегментов приводит к неэффективному использованию ОП. Так, если ОП заполнена, то при замещении одного из сегментов требуется вытеснить такой, размер которого равен или больше размера нового. При многократном повторе подобных действий в ОП остается множество свободных участков, недостаточных по размеру для загрузки полного сегмента. Решением проблемы служит *сегментно-страничная организация памяти*. В ней размер сегмента выбирается не произвольно, а задается кратным размеру страницы. Сегмент может содержать то или иное, но обязатель-

но целое число страниц, даже если одна из страниц заполнена частично. Возникает определенная иерархия в организации доступа к данным, состоящая из трех ступеней: сегмент > страница > слово. Этой структуре соответствует иерархия таблиц, служащих для перевода виртуальных адресов в физические. В сегментной таблице программы перечисляются все сегменты данной программы с указанием начальных адресов СТ, относящихся к каждому сегменту. Количество страничных таблиц равно числу сегментов и любая из них определяет расположение каждой из страниц сегмента в памяти, которые могут располагаться не подряд - часть страниц может находиться в ОП, остальные - во внешней памяти. Структуру виртуального адреса и процесс преобразования его в физический адрес иллюстрирует рис. 5.34.

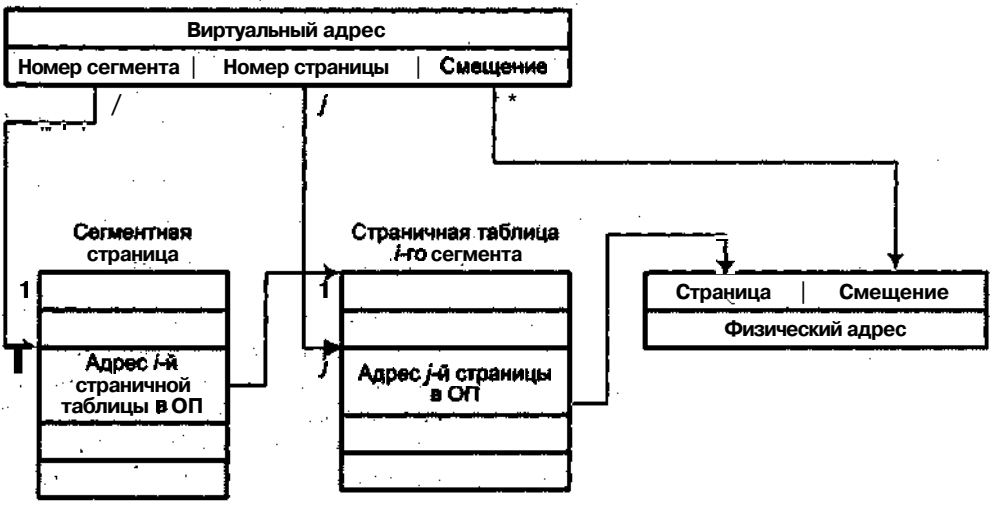


Рис. 5.34. Преобразование адреса при сегментно-страничной организации памяти

Для получения физического адреса необходим доступ к сегментной и одной из страничных таблиц, поэтому преобразование адреса может занимать много времени.

Организация защиты памяти

Современные вычислительные машины, как правило, работают в многопользовательском и многозадачном режимах, когда в основной памяти одновременно находятся программы, относящиеся как к разным пользователям, так и к различным задачам одного пользователя. Если даже ВМ выполняет только одну программу, в ОП, помимо этой программы и относящихся к ней данных, всегда присутствуют фрагменты операционной системы. Каждой задаче в основной памяти выделяется свое адресное пространство. Такие пространства, если только это специально не предусмотрено, обычно независимы. В то же время в программах могут содержаться ошибки, приводящие к вторжению в адресное пространство других задач. Следствием этих ошибок может стать искажение информации, принадлежащей другим программам. Следовательно, в ВМ обязательно должны быть предусмотрены меры,

предотвращающие несанкционированное воздействие программ одного пользователя на работу программ других пользователей и на операционную систему. Особенно опасны последствия таких ошибок при нарушении адресного пространства операционной системы.

Чтобы воспрепятствовать разрушению одних программ другими, достаточно защитить область памяти данной программы от попыток записи в него со стороны других программ (защита от записи). В ряде случаев необходимо иметь возможность защиты и от чтения со стороны других программ, например при ограничениях на доступ к системной информации.

Защита от вторжения программ в чужие адресные пространства реализуется различными средствами и способами, но в любом варианте к системе защиты предъявляются два требования: ее реализация не должна заметно снижать производительность ВМ и требовать слишком больших аппаратных затрат.

Задача обычно решается за счет сочетания программных и аппаратных средств, хотя ответственность за охрану адресных пространств от несанкционированного доступа обычно возлагается на операционную систему. В учебнике рассматриваются, главным образом, аппаратные аспекты проблемы защиты памяти.

Защита отдельных ячеек памяти

Этим видом защиты обычно пользуются при отладке новых программ параллельно с функционированием других программ. Реализовать подобный режим можно за счет выделения в каждой ячейке памяти специального «разряда защиты» и связывания его со схемой управления записью в память/Установка этого разряда в 1 блокирует запись в данную ячейку. Подобный режим использовался в вычислительных машинах предыдущих поколений (ДЛ Я современных ВМ он не типичен).

Кольца защиты

Защиту адресного пространства операционной системы от несанкционированного вторжения со стороны пользовательских программ обычно организуют за счет аппаратно реализованного разделения системного и пользовательского уровней привилегий. Предусматриваются как минимум два режима работы процессора: системный (режим супервизора - «надзирателя») и пользовательский. Такую структуру принято называть *кольцами защиты* и изображать в виде концентрических окружностей, где пользовательский режим представлен внешним кольцом, а системный — внутренней окружностью. В системном режиме программе доступны все ресурсы ВМ, а возможности пользовательского режима существенно ограничены. Переключение из пользовательского режима в системный осуществляется специальной командой. В большинстве современных ВМ число уровней привилегий (колец защиты) увеличено. Так, в микропроцессорах класса Pentium предусмотрено четыре уровня привилегий.

Метод граничных регистров

Данный вид защиты наиболее распространен. Метод предполагает наличие в процессоре двух *граничных регистров,* содержимое которых определяет нижнюю и верхнюю границы области памяти, куда программа имеет право доступа. Заполнение

граничных регистров производится операционной системой при загрузке программы. При каждом обращении к памяти проверяется, попадает ли используемый адрес в установленные границы. Такую проверку, например, можно организовать на этапе преобразования виртуального адреса в физический. При нарушении границы доступ к памяти блокируется, и формируется запрос прерывания, вызывающий соответствующую процедуру операционной системы. Нижнюю границу разрешенной области памяти определяет сегментный регистр. Верхняя граница подсчитывается операционной системой в соответствии с размером размещаемого в ОП сегмента.

В рассмотренной схеме необходимо, чтобы в ВМ поддерживались два режима работы: привилегированный и пользовательский. Запись информации в граничные регистры возможна лишь в привилегированном режиме.

Метод ключей защиты

Метод позволяет организовать защиту несмежных областей памяти. Память условно делится на блоки одинакового размера. Каждому блоку ставится в соответствие некоторый код, называемый *ключом защиты памяти.* Каждой программе, в свою очередь, присваивается *код защиты программы.* Условием доступа программы к конкретному блоку памяти служит совпадение ключей защиты памяти и программы, либо равенство одного из этих ключей нулю. Нулевое значение ключа защиты программы разрешает доступ ко всему адресному пространству и используется только программами операционной системы. Распределением ключей защиты программы ведает операционная система. Ключ защиты программы обычно представлен в виде отдельного поля *слова состояния программы,* хранящегося в специальном регистре. Ключи защиты памяти хранятся в специальной памяти. При каждом обращении к ОП специальная комбинационная схема производит сравнение ключей защиты памяти и программы. При совпадении доступ к памяти разрешается. Действия в случае несовпадения ключей зависят от того, какой вид доступа запрещен: при записи, при чтении или в обоих случаях. Если выяснилось, что данный вид доступа запрещен, то так же как и в методе граничных регистров формируется запрос прерывания и вызывается соответствующая процедура операционной системы.

Внешняя память

Важным звеном в иерархии запоминающих устройств является внешняя, или вторичная память, реализуемая на базе различных ЗУ. Наиболее распространенные виды таких ЗУ — это магнитные и оптические диски и магнитоленточные устройства.

Магнитные диски

Информация в ЗУ на магнитных дисках (МД) хранится на плоских металлических или пластиковых пластинах (дисках), покрытых магнитным материалом. Данные записываются и считываются с диска с помощью электромагнитной катушки» называемой *головкой считывания/записи,* которая в процессе считывания и записи

неподвижна, в то время как диск вращается относительно нее. При записи на головку подаются электрические импульсы, намагничивающие участок поверхности под ней, причем характер намагниченности поверхности различен в зависимости от направления тока в катушке. Считывание базируется на электрическом токе, наводимом в катушке головки, под воздействием перемещающегося относительно нее магнитного поля. Когда под головкой проходит участок поверхности диска, в катушке наводится ток той же полярности, что использовался для записи информации. Несмотря на разнообразие типов магнитных дисков, принципы их организации обычно однотипны.

Организация данных и форматирование

Данные на диске организованы в виде набора концентрических окружностей, называемых *дорожками* (рис. 5.35). Каждая из них имеет ту же ширину, что и головка. Соседние дорожки разделены промежутками. Это предотвращает ошибки из-за смещения головки или из-за интерференции магнитных полей. Как правило, для упрощения электроники принимается, что на всех дорожках может храниться одинаковое количество информации. Таким образом, плотность записи увеличивается от внешних дорожек к внутренним.

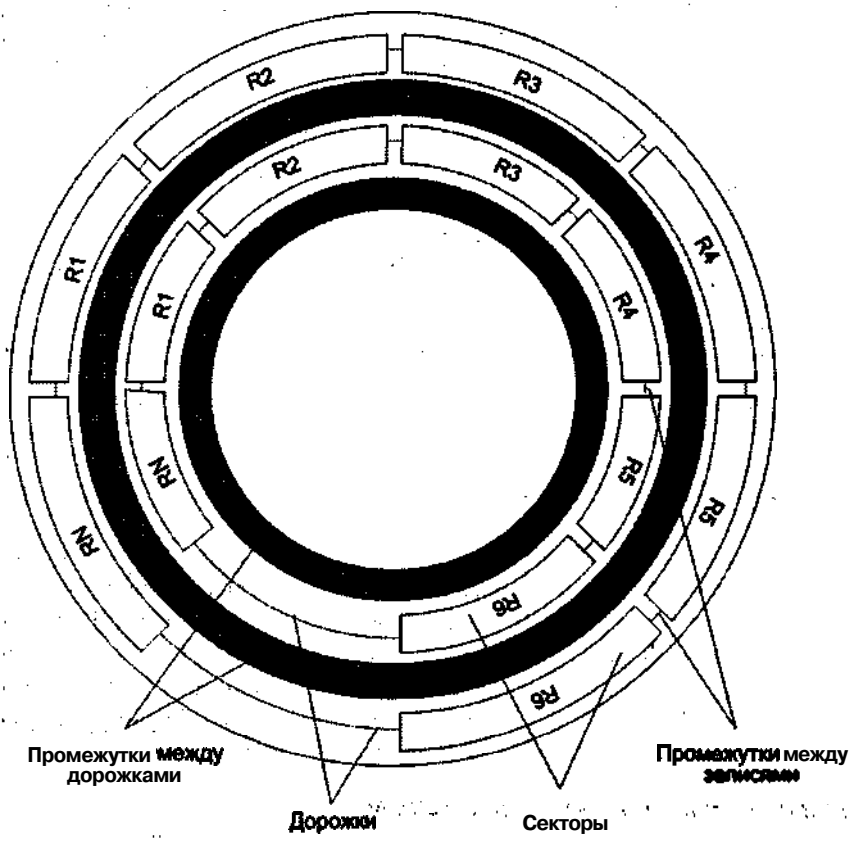


Рис. 5.35. Порядок размещения информации на магнитном диске

Обмен информацией с МД осуществляется блоками. Размер блока обычно меньше емкости дорожки, и данные на дорожке хранятся в виде последовательных областей — *секторов,* разделенных между собой промежутками. Размер сектора равен минимальному размеру блока.

Типовое число секторов на дорожке колеблется от 10 до 100. При такой организации должны быть заданы точка отсчета секторов и способ определения начала и конца каждого сектора. Все это обеспечивается с помощью *форматирования,* в ходе которого на диск заносится служебная информация, недоступная пользователю и используемая только аппаратурой дискового ЗУ.

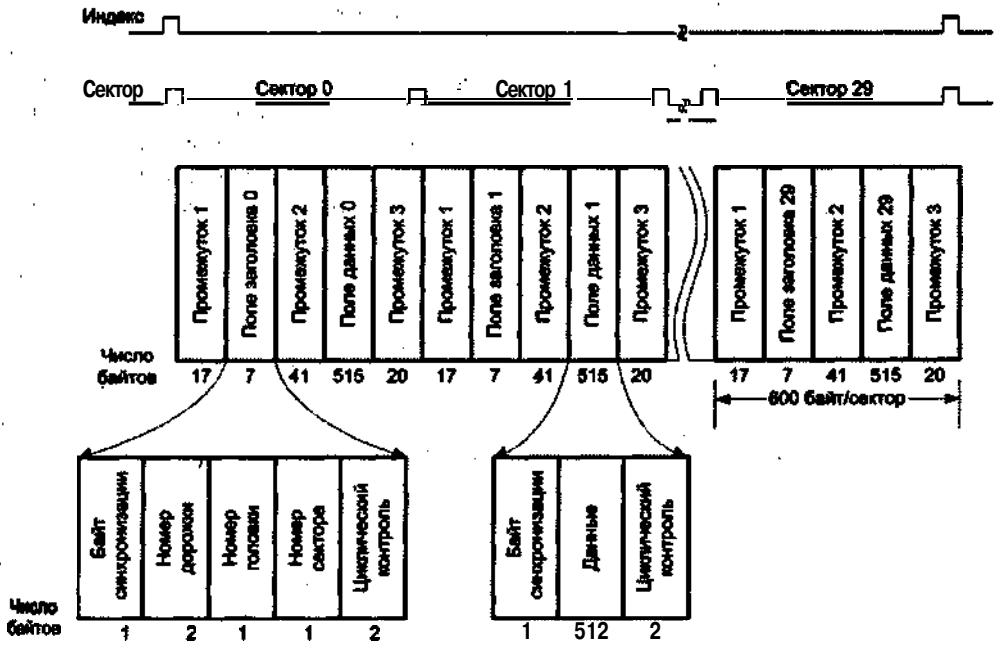


Рис. 5.36. Формат дорожки диска типа «Винчестер» (Seagate ST506)

Пример разметки МД показан на рис. 5.36. Здесь каждая дорожка включает в себя 30 секторов по 600 байт в каждом. Сектор хранит 512 байт данных и управляющую информацию, нужную для контроллера диска. Поле заголовка содержит информацию, служащую для идентификации сектора. Байт синхронизации представляет собой характерную двоичную комбинацию, позволяющую определить начало поля. Номер дорожки определяет дорожку на поверхности. Если в накопителе используется несколько дисков, то номер головки определяет нужную поверхность. Поле заголовка и поле данных содержат также код циклического контроля, позволяющий обнаружить ошибки. Обычно этот код формируется последовательным сложением по модулю 2 всех байтов, хранящихся в поле.

Характеристики дисковых систем

В табл. 5.8 приведена классификация разнообразных систем МД.

Таблица 5.8. Характеристики дисковых систем

Движение головок	**Число пластин**
Фиксированные Подвижные	Однодисковые Многодисковые
Сменяемость дисков	**Механизм головки**
Несъемные диски Съемные диски	**Контактный** **С фиксированным зазором** **С аэродинамическим зазором**
Использование поверхностей	
Односторонние Двухсторонние	

В ЗУ с *фиксированными головками* приходится по одной головке считывания/записи на каждую дорожку. Головки смонтированы на жестком рычаге, пересекающем все дорожки диска (рис. 5.37, *а).* В дисковом ЗУ с *подвижными головками* имеется только одна головка, также установленная на рычаге (рис. 5.37,*б),* однако рычаг способен перемещаться в радиальном направлении над поверхностью диска, обеспечивая возможность позиционирования головки на любую дорожку.

Диски с магнитным носителем устанавливаются в дисковод, состоящий из рычага, шпинделя, вращающего диск, и электронных схем, требуемых для ввода и вывода двоичных данных. *Несъемный диск* зафиксирован на дисководе. *Съемный дис* может быть вынут из дисковода и заменен на другой аналогичный диск. Преимущество системы со съемными дисками — возможность хранения неограниченного объема данных при ограниченном числе дисковых устройств. Кроме того, такой диск может быть перенесен с одной ВМ на другую.

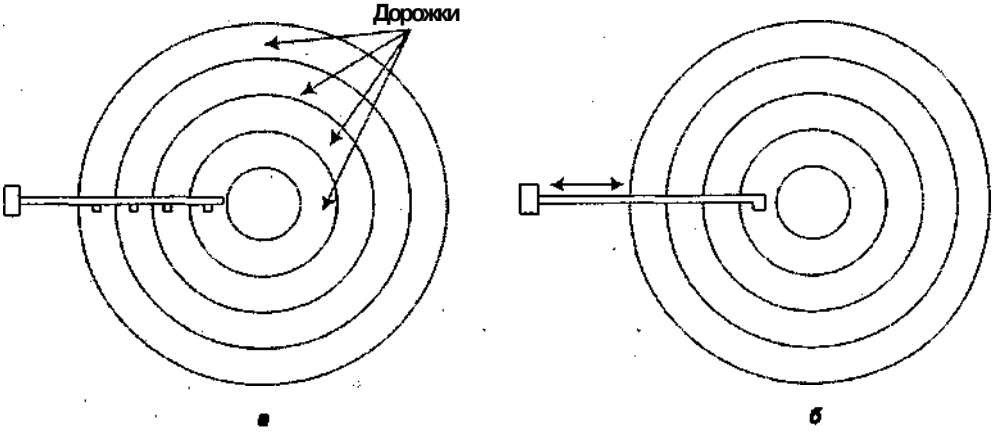


Рис. 5.37. Варианты организации дисков: а - с фиксированными головками; *б* - с подвижной головкой

Большинство дисков имеет магнитное покрытие с обеих сторон. В этом случае говорят о *двусторонних* (double-sided) дисках. *Односторонние* (single-sided) диски в наше время встречаются достаточно редко.

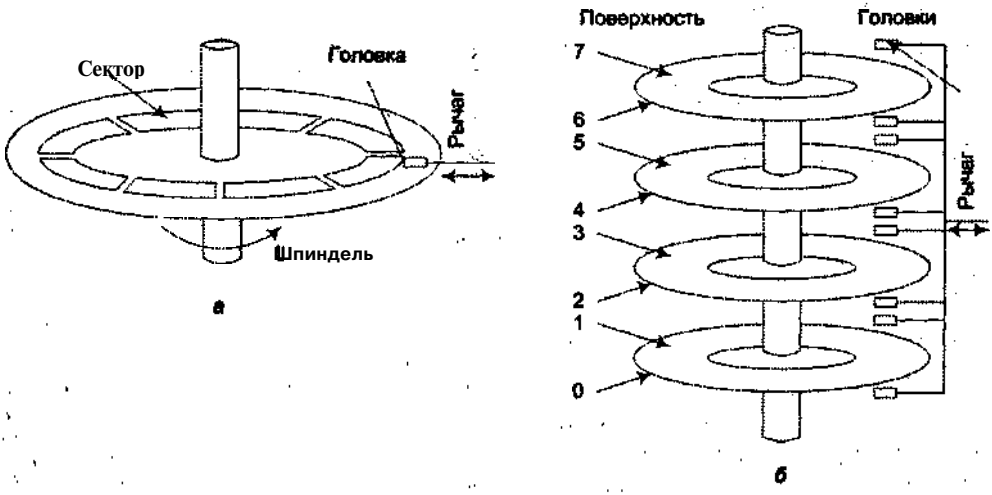


Рис. 5.38. Дисковое ЗУ: а - с одной пластиной; *б* - с дисковым пакетом

На оси может располагаться один (рис. 5.38, *а)* или несколько (рис. 5.38, *б)* дисков. В последнем случае используют термин *дисковый пакет*.

В зависимости от применяемой головки считывания/записи можно выделить три типа дисковых ЗУ. В первом варианте головка устанавливается на фиксированной дистанции от поверхности так, чтобы между ними оставался воздушный промежуток. Второй вариант - это когда в процессе чтения и записи головка и диск находятся в физическом контакте. Такой способ используется, например, в накопителях на гибких магнитных дисках (дискетах).

Для правильной записи и считывания головка должна генерировать и воспринимать магнитное поле определенной интенсивности, поэтому чем уже головка, тем ближе должна она размещаться к поверхности диска (более узкая головка означает и более узкие дорожки, а значит, и большую емкость диска). Однако приближение головки к диску означает и больший риск ошибки за счет загрязнения и дефектов. В процессе решения этой проблемы был создан диск типа *винчестер*. Головки винчестера и диски помещены в герметичный корпус, защищающий их от загрязнения. Кроме того, головки имеют очень малый вес и аэродинамическую форму. Они спроектированы для работы значительно ближе к поверхности диска, чем головки в обычных жестких дисках, тем самым повышается плотность хранения данных. При остановленном диске головка прилегает к его поверхности. Давления, возникающего при вращении диска, достаточно для подъема головки над Поверхностью. В результате создается неконтактная система с узкими головками считывания/записи, обеспечивающая более плотное прилегание головки к поверхности диска, чем в обычных жестких дисках.

Массивы магнитных дисков с избыточностью

Магнитные диски, будучи основой внешней памяти любой ВМ, одновременно остаются и одним из «узких мест» из-за сравнительно высокой стоимости, недостаточной производительности и отказоустойчивости. Характерно, что если в плане

стоимости и надежности ситуация улучшается, то разрыв в производительности между МД и ядром ВМ постоянно растет. Так, при удвоении быстродействия процессоров примерно каждые два года для МД такое удвоение было достигнуто лишь спустя десять лет. Ясно, что уже с самого начала использования подсистем памяти на базе МД не прекращаются попытки улучшить их характеристики. Одно из наиболее интересных и универсальных усовершенствований было предложено в 1987 году учеными университета Беркли (Калифорния) [179]. Проект известен под аббревиатурой RAID (Redundant Array of Independent (or Inexpensive) Disks) - *массив независимых (или недорогих) дисков с избыточностью*. В основе концепции RAID лежит переход от одного физического МД большой емкости к массиву недорогих, независимо и параллельно работающих физических дисковых ЗУ, рассматриваемых операционной системой как одно большое логическое дисковое запоминающее устройство. Такой подход позволяет повысить производительность дисковой памяти за счет возможности параллельного обслуживания запросов на считывание и запись, при условии, что данные находятся на разных дисках. Повышенная надежность достигается тем, что в массиве дисков хранится избыточная информация, позволяющая обнаружить и исправить возможные ошибки. На период, когда концепция RAID была впервые предложена, определенный выигрыш достигался и в плане стоимости. В настоящее время, с развитием технологии производства МД, утверждение об экономичности массивов RAID становится проблематичным, что, однако, вполне компенсируется их повышенными быстродействием и отказоустойчивостью.

В [179] рассмотрены пять схем организации данных и способов введения избыточности, названные авторами уровнями RAID: RAID 1, RAID 2,..., RAID 5. В настоящее время производители RAID-систем, объединившиеся в ассоциацию RAB (RAID Advisory Board), договорились о единой классификации RAID, включающей в себя шесть уровней (добавлен уровень RAID 0). Известны также еще несколько схем RAID, не включенных в эту классификацию, поскольку по сути они представляют собой различные комбинации стандартных уровней. Хотя ни одна из схем массива МД не может быть признана идеальной для всех случаев, каждая из них позволяет существенно улучшить какой-то из показателей (производительность, отказоустойчивость) либо добиться наиболее подходящего сочетания этих показателей. Для всех уровней RAID характерны три общих свойства:

- RAID представляет собой набор физических дисковых ЗУ, управляемых операционной системой и рассматриваемых как один логический диск;
- данные распределены по физическим дискам массива;
- избыточное дисковое пространство используется для хранения дополнительной информации, гарантирующей восстановление данных в случае отказа диска

Повышение производительности дисковой подсистемы

Повышение производительности дисковой подсистемы в RAID достигается с помощью приема, *называемого расслоением или расщеплением* (striping). В его основе лежит разбиение данных и дискового пространства на сегменты, так называемые *полосы* (strip - узкая полоса). Полосы распределяются по различным дискам массива, в соответствии с определенной системой. Это позволяет производить па-

раллельное считывание или запись сразу нескольких полос, если они расположены на разных дисках. В идеальном случае производительность дисковой подсистемы может быть увеличена в число раз, равное количеству дисков в массиве. Размер (ширина) полосы выбирается исходя из особенностей каждого уровня RAID и может быть равен биту, байту, размеру физического сектора МД (обычно 512 байт) или размеру дорожки.

Чаще всего логически последовательные полосы распределяются по последовательным дискам массива. Так, в *n*-дисковом массиве *n* первых логических полос физически расположены как первые полосы на каждом из *n* дисков, следующие *n* полос — как вторые полосы на каждом физическом диске и т. д. Набор логически последовательных полос, одинаково расположенных на каждом ЗУ **массива,** называют *поясом* (stripe - широкая полоса).

Как уже упоминалось, минимальный объем информации, считываемый с МД или записываемый на него за один раз, равен размеру физического сектора диска. Это приводит к определенным проблемам при меньшей ширине полосы, которые RAID обычно решаются за счет усложнения контроллера МД.

Повышение отказоустойчивости дисковой подсистемы

Одной из целей концепции RAID была возможность обнаружения и коррекции ошибок, возникающих при отказах дисков или в результате сбоев. Достигается это за счет избыточного дискового пространства, которое задействуется для хранения дополнительной информации, позволяющей восстановить искаженные или утерянные данные. В RAID предусмотрены три вида такой информации:

- дублирование;
- код Хэмминга;
- биты паритета.

Первый из вариантов заключается в дублировании всех данных, при условии, что экземпляры одних и тех же данных расположены на разных дисках массива. Это позволяет при отказе одного из дисков воспользоваться соответствующей информацией, хранящейся на исправных МД. В принципе распределение информации по дискам массива может быть произвольным, но для сокращения издержек, связанных с поиском копии, обычно применяется разбиение массива на пары МД, где в каждой паре дисков информация идентична и одинаково расположена. При таком дублировании для управления парой дисков может использоваться общий или раздельные контроллеры. Избыточность дискового массива здесь составляет 100%.

Второй способ формирования корректирующей информации основан на вычислении кода Хэмминга для каждой группы полос, одинаково расположенных на всех дисках массива (пояса). Корректирующие биты хранятся на специально выделенных для этой цели дополнительных дисках (по одному диску на каждый бит). Так, для массива из десяти МД требуются четыре таких дополнительных диска, и избыточность в данном случае близка к 30%.

В третьем случае вместо кода Хэмминга для каждого набора полос, расположенных в идентичной позиции на всех дисках массива, вычисляется контрольная полоса, состоящая из битов паритета. В ней значение отдельного бита формируется как сумма по модулю два для одноименных битов во всех контролируемых по-

лосах. Для хранения полос паритета требуется только один дополнительный диск. В случае отказа какого-либо из дисков массива производится обращение к диску паритета, и данные восстанавливаются по битам паритета и данным от остальных дисков массива. Реконструкция данных достаточно проста. Рассмотрим массив из пяти дисковых ЗУ, где диски *XO-X3* содержат данные, а *X4* — это диск паритета. Паритет для /-го бита вычисляется как

$$X4_i - X3_i \Phi X2_i \Phi X1_i \Phi X0_i.$$

Предположим, что дисковод *X1* отказал. Если мы добавим $X4_i \Phi X1_i$ к обеим частям предыдущего выражения, то получим:

$$X1_i = X4_i \oplus X3_i \oplus X2_i \oplus X0_i.$$

Таким образом, содержимое каждой полосы данных на любом диске массива может быть восстановлено по содержимому соответствующих полос на остальных дисках массива. Избыточность при таком способе в среднем близка к 20%.

RAID уровня О

RAID уровня 0, строго говоря, не является полноценным членом семейства RAID, поскольку данная схема не содержит избыточности и нацелена только на повышение производительности в ущерб надежности.

В основе RAID 0 лежит расслоение данных. Полосы распределены по всем дискам массива дисковых ЗУ по циклической схеме (рис. 5.39). Преимущество такого распределения в том, что если требуется записать или прочитать логически последовательные полосы, то несколько таких полос (вплоть до *n)* могут обрабатываться параллельно, за счет чего существенно снижается общее время ввода/вывода. Ширина полос в RAID 0 варьируется в зависимости от применения, но в любом случае она не менее размера физического сектора МД.

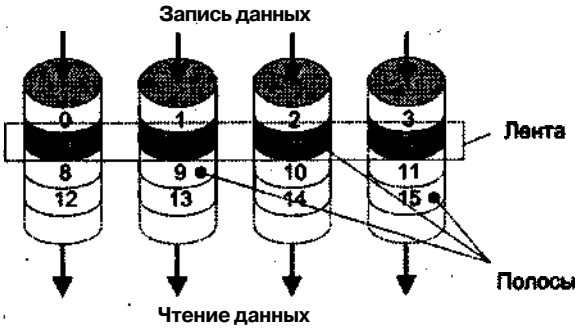


Рис. 5.39. RAID уровня О

RAID 0 обеспечивает наиболее эффективное использование дискового пространства и максимальную производительность дисковой подсистемы при минимальных затратах и простоте реализации. Недостатком является незащищенность данных — отказ одного из дисков ведет к разрушению целостности данных во всем массиве. Тем не менее существует ряд приложений, где производительность и емкость дисковой системы намного важнее возможного снижения надежности. К таким можно отнести задачи, оперирующие большими файлами данных, в основном

в режиме считывания информации (библиотеки изображений, большие таблицы и т. п.), и где загрузка информации в основную память должна производиться как можно быстрее. Учитывая отсутствие в RAID G средств по защите данных, желательно хранить дубликаты файлов на другом, более надежном носителе информации, например на магнитной ленте.

RAID уровня 1

В RAID 1 избыточность достигается с помощью дублирования данных. В принципе исходные данные и их копии могут размещаться по дисковому массиву произвольно, главное чтобы они находились на разных дисках. В плане быстродействия и простоты реализации выгоднее, когда данные и копии располагаются идентично на одинаковых дисках. Рисунок 5.40 показывает, что, как и в RAID 0, здесь имеет место разбиение данных на полосы. Однако в этом случае каждая логическая полоса отображается на два отдельных физических диска, так что каждый диск в массиве имеет так называемый «зеркальный» диск, содержащий идентичные данные.

Для управления каждой парой дисков может быть использован общий контроллер, тогда данные сначала записываются на основной диск, а затем. — на «зеркальный» («зеркалирование»). Более эффективно применение самостоятельных контроллеров для каждого диска, что позволяет производить одновременную запись на оба диска.

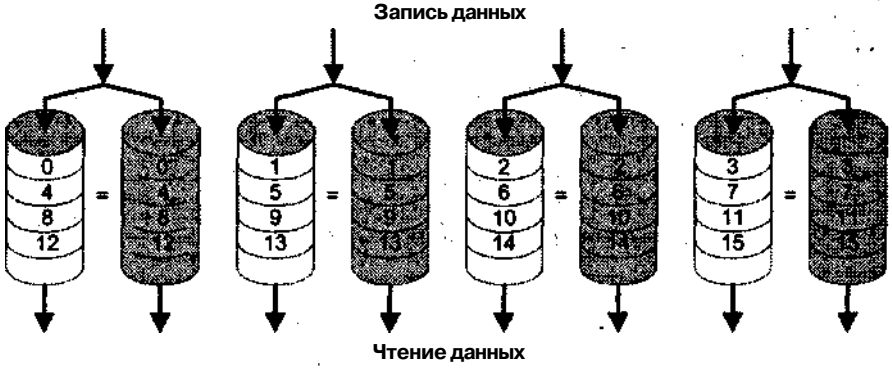


Рис. 5.40. RAID уровня 1

Запрос на чтение может быть обслужен тем из двух дисков, которому в данный момент требуется меньшее время поиска и меньшая задержка вращения. Запрос на запись требует, чтобы были обновлены обе соответствующие полосы, но это выполнимо и параллельно, причем задержка определяется тем диском, которому нужны большие время поиска и задержка вращения. В то же время у RAID 1 нет дополнительных затрат времени на вычисление вспомогательной корректирующей информации. Когда одно дисковое ЗУ отказывает, данные могут быть просто взяты со второго.

Принципиальный изъян RAID 1 - высокая стоимость: требуется вдвое больше физического дискового пространства. По этой причине использование RAID 1 обычно ограничивают хранением загрузочных разделов, системного программного

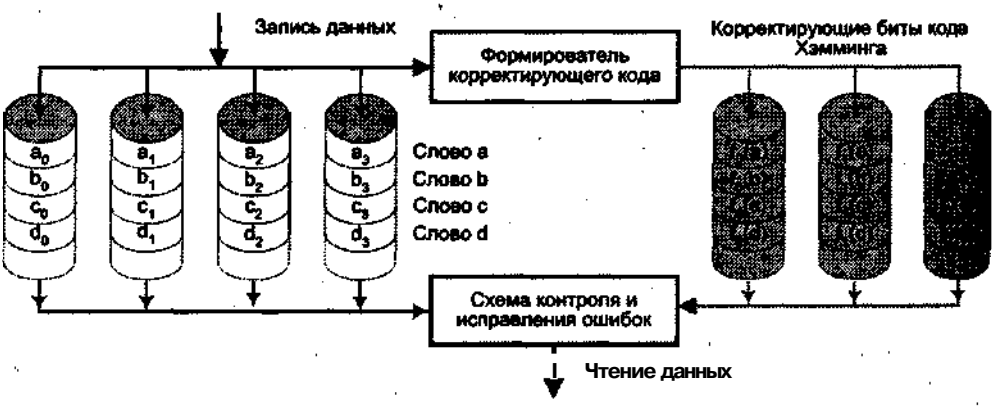
обеспечения и данных, а также других особенно критичных файлов: RAID 1 обеспечивает резервное копирование всех данных, так что в случае отказа диска критическая информация доступна практически немедленно.

RAID уровня 2

В системах RAID 2 используется техника параллельного доступа, где в выполнении каждого запроса на В/ВЫВ одновременно участвуют все диски. Обычно шпиндели всех дисков синхронизированы так, что головки каждого ЗУ в каждый момент времени находятся в одинаковых позициях. Данные разбиваются на полосы длиной в 1 бит и распределены по дискам массива таким образом, что полное машинное слово представляется поясом, то есть число дисков равно длине машинного слова в битах. Для каждого слова вычисляется корректирующий код (обычно это код Хэмминга, способный корректировать одиночные и обнаруживать двойные ошибки), который, также побитово, хранится на дополнительных дисках (рис. 5.41). Например, для массива, ориентированного на 32-разрядные слова (32 основных диска) требуется семь дополнительных дисковых ЗУ (корректирующий код состоит из 7 разрядов).

При записи вычисляется корректирующий код, который заносится на отведенные для него диски. При каждом чтении производится доступ ко всем дискам массива, включая дополнительные. Считанные данные вместе с корректирующим кодом подаются на контроллер дискового массива, где происходит повторное вычисление корректирующего кода и его сравнение с хранившимся на избыточных дисках. Если присутствует одиночная ошибка, контроллер способен ее мгновенно распознать и исправить, так что время считывания не увеличивается.

RAID 2 позволяет достичь высокой скорости В/ВЫВ при работе с большими последовательными записями, но становится неэффективным при обслуживании записей небольшой длины. Основное преимущество RAID 2 состоит в высокой степени защиты информации, однако предлагаемый в этой схеме метод коррекции уже встроен в каждое из современных дисковых ЗУ.



ких избыточных дисков представляется неэффективным, и массивы уровня RAID 2 в настоящее время не производятся.

RAID уровня 3

RAID 3 организован сходно с RAID2. Отличие в том, что RAID 3 требует только одного дрполнительного диска — диска паритета, вне зависимости от того, насколько велик массив дисков (рис. 5.42). В RAID 3 используется параллельный доступ к данным, разбитым на полосы длиной в бит или байт. Все диски массива синхронизированы. Вместо кода Хэмминга для набора полос идентичной позиции на всех дисках массива (пояса) вычисляется полоса, состоящая из битов паритета. В случае отказа дискового ЗУ производится обращение к диску паритета, и данные восстанавливаются по битам паритета и данным от остальных дисков массива.

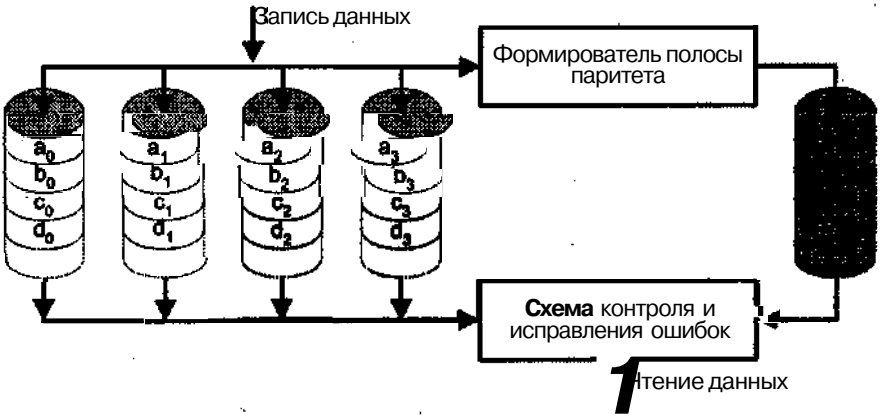


Рис. 5.42. RAID уровня 3

Так как данные разбиты на очень маленькие полосы, RAID 3 позволяет достигать очень высоких скоростей передачи данных. Каждый запрос на ввод/вывод приводит к параллельной передаче данных со всех дисков. Для приложений, связанных с большими пересылками данных, это обстоятельство очень существенно. С другой стороны, параллельное обслуживание одиночных запросов невозможно, и производительность дисковой подсистемы в этом случае падает.

Ввиду того что для хранения избыточной информации нужен всего один диск, причем независимо от их числа в массиве, именно уровню RAID 3 отдается предпочтение перед RAID 2.

RAID уровня 4

По своей идее и технике формирования избыточной информации RAID 4 идентичен RAID 3, только размер полос в RAID 4 значительно больше (обычно один-два физических блока на диске). Главное отличие состоит в том, что в RAID 4 используется техника независимого доступа, когда каждое ЗУ массива в состоянии функционировать независимо, так, что отдельные запросы на ввод/вывод могут удовлетворяться параллельно (рис. 5.43).

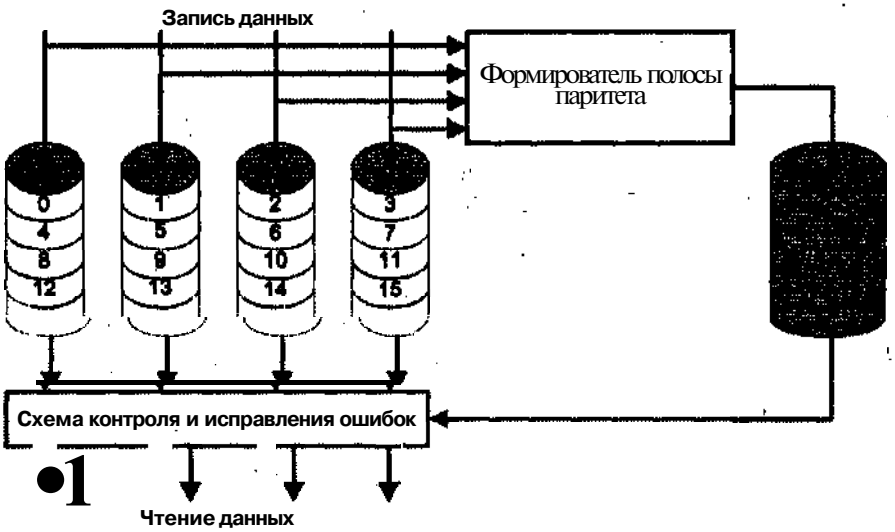


Рис. 5.43. RAID уровня 4

Для RAID 4 характерны издержки, обусловленные независимостью дисков. Если в RAID 3 запись производилась одновременно для всех полос одного пояса, в RAID 4 осуществляется запись полос в разные пояса. Это различие ощущается особенно при записи данных малого размера.

Каждый раз для выполнения записи программное обеспечение дискового массива должно обновить не только данные пользователя, но и соответствующие биты паритета. Рассмотрим массив из пяти дисковых ЗУ, где ЗУ X0... X3 содержат данные, а X4 - диск паритета. Положим, что производится запись, охватывающая только полосу на диске X1. Первоначально для каждого бита *i* мы имеем следующее соотношение:

$$X4_i = X3_i \oplus X2_i \oplus X1_i \oplus X0_i.$$

После обновления для потенциально измененных битов, обозначаемых с помощью апострофа, получаем:

$$X4_i' = X3_i \oplus X2_i \oplus X1_i' \oplus X0_i =$$
$$= X3_i \oplus X2_i \oplus x \setminus; \oplus X0_i \oplus X1_i \oplus X1_i -$$
$$- X4_i \oplus X1_i \oplus X1_i'.$$

Для вычисления новой полосы паритета программное обеспечение управления массивом должно прочитать старую полосу пользователя и старую полосу паритета. Затем оно может заменить эти две полосы новой полосой данных и новой вычисленной полосой паритета. Таким образом, запись каждой полосы связана с двумя операциями чтения и двумя операциями записи.

В случае записи большого объема информации, охватывающего полосы на всех дисках, паритет вычисляется достаточно легко путем расчета, в котором участвуют только новые биты данных, то есть содержимое диска паритета может быть обновлено параллельно с дисками данных и не требует дополнительных операций чтения и записи.

Массивы RAID 4 наиболее подходят для приложений, требующих поддержки высокого темпа поступления запросов ввода/вывода, и уступает RAID 3 там, где приоритетен большой объем пересылок данных.

RAID уровня 5

RAID 5 имеет структуру, напоминающую RAID 4. Различие заключается в том, что RAID 5 не содержит отдельного диска для хранения полос паритета, а разносит их по всем дискам. Типичное распределение осуществляется по циклической схеме, как это показано на рис. 5.44. В *n*-дисковом массиве полоса паритета вычисляется для полос *n-1* дисков, расположенных в одном поясе, и хранится в том же поясе, но на диске, который не учитывался при вычислении паритета. При переходе от одного пояса к другому эта схема циклически повторяется.

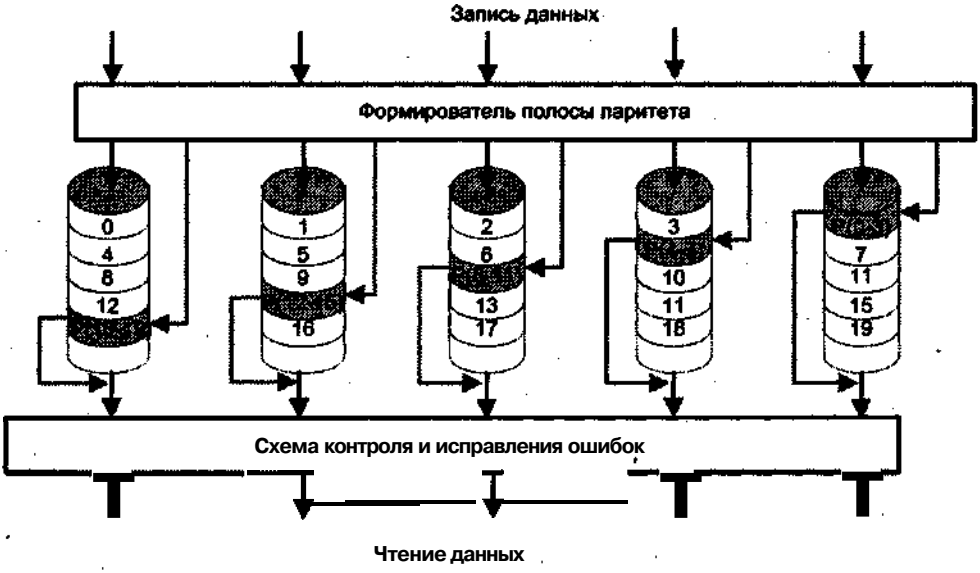


Рис. 5.44. RAID уровня 5

Распределение полос паритета по всем дискам предотвращает возникновение проблемы, упоминавшейся для RAID 4.

RAID уровня 6

RAID 6 очень похож на RAID 5. Данные также разбиваются на полосы размером в блок и распределяются по всем дискам массива. Аналогично, полосы паритета распределены по разным дискам. Доступ к полосам независимый и асинхронный. Различие состоит в том, что на каждом диске хранится не одна, а две полосы паритета. Первая из них, как и в RAID 5, содержит контрольную информацию для полос, расположенных на горизонтальном срезе массива (за исключением диска, где эта полоса паритета хранится). В дополнение формируется и записывается вторая полоса паритета, контролирующая все полосы какого-то одного диска массива (вертикальный срез массива), но только не того, где хранится полоса паритета. Сказанное иллюстрируется рис. 5.45.

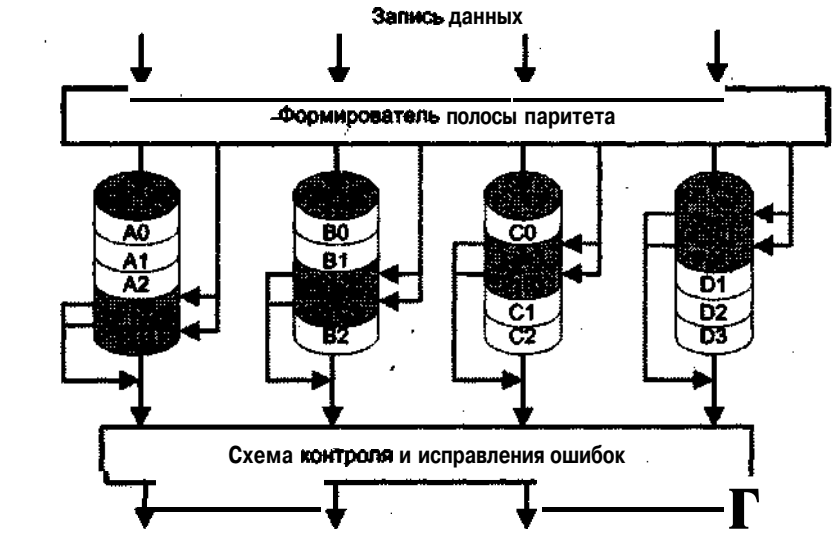


Рис. 5.45. RAID уровня 6

Такая схема массива позволяет восстановить информацию при отказе сразу двух дисков. С другой стороны, увеличивается время на вычисление и запись паритетной информации и требуется дополнительное дисковое пространство. Кроме того, реализация данной схемы связана с усложнением контроллера дискового массива. В силу этих причин схема среди выпускаемых RAID-систем встречается крайне редко.

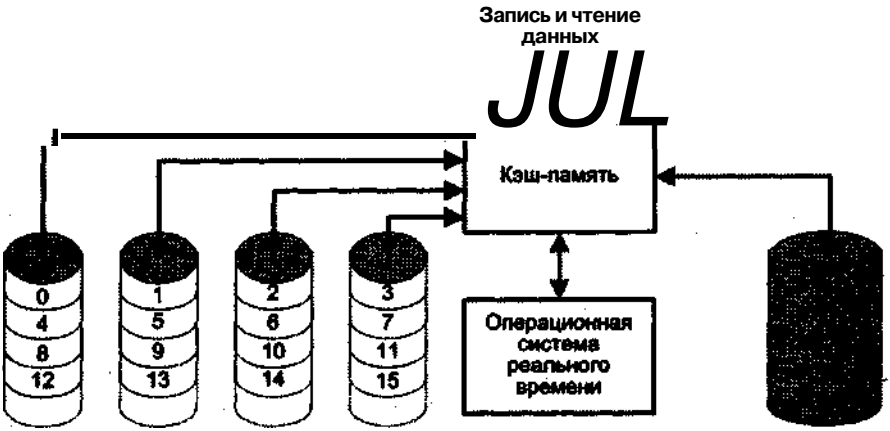


Рис. 5.46. RAID уровня 7

RAID уровня 7

Схема RAID 7, запатентованная Storage Computer Corporation, объединяет массив асинхронно работающих дисков и кэш-память, управляемые встроенной в кон-

троллер массива операционной системой реального времени (рис. 5.46). Данные разбиты на полосы размером в блок и распределены по дискам массива. Полосы паритета хранятся на специально выделенных для этой цели одном или нескольких дисках.

Схема некритична к виду решаемых задач и при работе с большими файлами не уступает по производительности RAID 3. Вместе с тем RAID 7 может так же эффективно, как и RAID 5, производить одновременно несколько операций чтения и записи для небольших объемов данных. Все это обеспечивается использованием кэш-памяти и собственной операционной системой.

RAID уровня 10

Данная схема совпадает с RAID 0, но в отличие от нее роль отдельных дисков выполняют дисковые массивы, построенные по схеме RAID 1 (рис. 5.47).

Таким образом, в RAID 10 сочетаются расслоение и дублирование. Это позволяет добиться высокой производительности, характерной для RAID 0 при уровне отказоустойчивости RAID 1. Основной недостаток схемы - высокая стоимость ее реализации. Кроме того, необходимость синхронизации всех дисков приводит к усложнению контроллера.

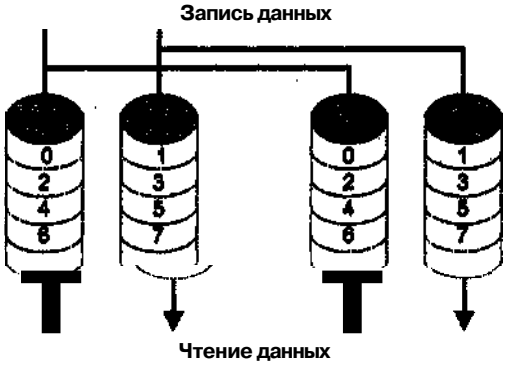


Рис. 5.47. RAID уровня 10

RAID уровня 53

В этом уровне сочетаются технологии RAID 0 и RAID 3, поэтому его правильнее было бы назвать RAID 30. В целом данная схема соответствует RAID 0, где роль отдельных дисков выполняют дисковые массивы, организованные по схеме RAID 3, Естественно, что в RAID 53 сочетаются все достоинства RAID 0 и RAID 3. Недостатки схемы такие же, что и у RAID 10.

Особенности реализации RAID-систем

Массивы RAID могут быть реализованы программно, аппаратно или как комбинация программных и аппаратных средств.

При программной реализации используются обычные дисководе контроллеры и стандартные команды ввода/вывода. Работа дисковых ЗУ в соответствии с алгоритмами различных уровней RAID обеспечивается программами операционной

системы ВМ. Программный режим RAID предусмотрен, например, в Windows NT. Это дает возможность программного изменения уровня RAID, в зависимости от особенностей решаемой задачи. Хотя программный способ является наиболее дешевым, он не позволяет добиться высокого уровня производительности, характерного для аппаратурной реализации RAID.

Аппаратурная реализация RAID предполагает возложение всех или большей части функций по управлению массивом дисковых ЗУ на соответствующее оборудование, при этом возможны два подхода. Первый из них заключается в замене стандартных контроллеров дисковых ЗУ на специализированные, устанавливаемые на место стандартных. Базовая ВМ общается с контроллерами на уровне обычных команд ввода/вывода, а режим RAID обеспечивают контроллеры. Как и обычные, специализированные контроллеры/адаптеры ориентированы на определенный вид шины. Поскольку наиболее распространенной шиной для подключения дисковых ЗУ в настоящее время является шина SCSI, большинство производителей RAID-систем ориентируют свои изделия на протокол SCSI, определяемый стандартами ANSI X3.131 и ISO/IEC. При втором способе аппаратной реализации RAID-система выполняется как автономное устройство, объединяющее в одном корпусе массив дисков и контроллер. Контроллер содержит микропроцессор и работает под управлением собственной операционной системы, полностью реализующей различные RAID-режимы. Такая подсистема подключается к шине базовой ВМ или к ее каналу ввода/вывода как обычное дисковое ЗУ.

При аппаратной реализации RAID-систем обычно предусматривается возможность замены неисправных дисков без потери информации и без остановки работы. Кроме того, многие из таких систем позволяют разбивать отдельные диски на разделы, причем разные разделы дисков могут объединяться в соответствии с различными уровнями RAID.

Оптическая память

В 1983 году была представлена первая цифровая аудиосистема на базе компакт-дисков (CD - compact disk). Компакт-диск - это односторонний диск, способный хранить более чем 60-минутную аудиоинформацию. Громадный коммерческий успех CD способствовал развитию технологии дешевых оптических запоминающих устройств для ВМ. За последующие годы были созданы различные системы памяти на оптических дисках, три из которых в прогрессирующей степени приживаются в вычислительных машинах: CD-ROM, WARM и стираемые оптические диски.

CD-ROM

Для аудио компакт-дисков и CD-ROM используется идентичная технология. Основное отличие состоит в том, что проигрыватели CD-ROM более прочные и содержат устройства для исправления ошибок, обеспечивающие корректность передачи данных с диска в ВМ. Диск изготавливается из пластмассы, например поликарбоната, и покрыт окрашенным слоем с высокой отражающей способностью, обычно алюминием. Цифровая информация заносится в виде микроскопических углублений в отражающей поверхности. Запись информации производится с помощью сильно сфокусированного луча лазера высокой интенсивности. Так созда-

ется так называемый мастер-диск, с которого затем печатаются копии. Углубления на копии защищаются от пыли и повреждений путем покрытия поверхности диска прозрачным лаком.

Информация с диска считывается маломощным лазером, расположенным в проигрывателе. Лазер освещает поверхность вращающегося диска сквозь прозрачное покрытие. Интенсивность отраженного луча лазера меняется, когда он попадает в углубление на диске. Эти изменения фиксируются фотодетектором и преобразуются в цифровой сигнал.

Углубления, расположенные ближе к центру диска, перемещаются относительно луча лазера медленнее, чем более удаленные. Из-за этого необходимы меры для компенсации различий в скорости так, чтобы лазер мог считывать информацию с постоянной скоростью.

Одно из возможных решений аналогично применяемому в магнитных дисках - увеличение расстояния между битами информации, в зависимости от ее расположения на диске. В этом случае диск может вращаться с неизменной скоростью и, соответственно, такие дисковые ЗУ известны как устройства *с постоянной угловой скоростью* (CAV, Constant Angular Velocity). Ввиду нерационального использования внешней части диска метод постоянной угловой скорости в CD-ROM не поддерживается. Вместо этого информация по диску размещается в секторах одинакового размера, которые сканируются с постоянной скоростью за счет того, что диск вращается с переменной скоростью. В результате углубления считываются лазером с *постоянной линейной скоростью* (CLV, Constant Linear Velocity). При доступе к информации у внешнего края диска скорость вращения меньше и возрастает при приближении к оси. Емкость дорожки и задержки вращения возрастают по мере смещения от центра к внешнему краю диска.

Выпускаются CD различной емкости. В типовом варианте расстояние между дорожками составляет 1,6 мк, что, с учетом промежутков между дорожками, позволяет обеспечить 20 344 дорожки. Фактически же, вместо множества концентрических дорожек, имеется одна дорожка в виде спирали, длина которой равна 5,27 км. Постоянная линейная скорость CD-ROM - 1 ,2 м/с, то есть для «прохождения» спирали требуется 4391 с или 73,2 мин. Именно эта величина составляет стандартное максимальное время проигрывания аудиодиска[1]. Так как данные считываются с диска со скоростью 176,4 Кбайт/с, емкость CD равна 774,57 Мбайт.

Данные на CD-ROM организованы как последовательность блоков. Типичный формат блока показан на рис. 5.48. Блок включает в себя следующие поля:

- **Синхронизация.** Это поле идентифицирует начало блока и состоит из нулевого байта, десяти байтов, содержащих только единичные разряды, и вновь байта из всех нулей.

[1] Строго определенная емкость компакт-дисков связана с интересной историей. Исполнительный директор фирмы Sony Акио Морита решил, что компакт-диски должны отвечать запросам исключительно любителей классической музыки, не более и не менее. После того как группа разработчиков провела опрос, выяснилось, что самым популярным классическим произведением в Японии в те времена была 9-я симфония Бетховена, которая длилась 72-73 минуты. Поэтому было решено, что компакт-диск должен быть рассчитан именно на 74 минуты звучания, а точнее на 74 мин и 33 с (стандарт, известный как «Красная Книга» - Red Book). Когда минуты пересчитали в килобайты, получилось 640 Мбайт. - *Примеч. лит. ред.*

Рис. 5.48. Формат блока CD-ROM

- **Идентификатор.** Заголовок, содержащий адрес блока и байт режима. Режим 0 определяет пустое поле данных; режим 1 отвечает за использование кода, корректирующего ошибки, и наличие 2048 байт данных; режим 2 определяет наличие 2336 байт данных и отсутствие корректирующего кода.
- **Данные.** Данные пользователя.
- **Корректирующий код (КК).** Поле предназначено для хранения дополнительных данных пользователя в режиме 2, а в режиме 1 содержит 288 байт кода с исправлением ошибок.

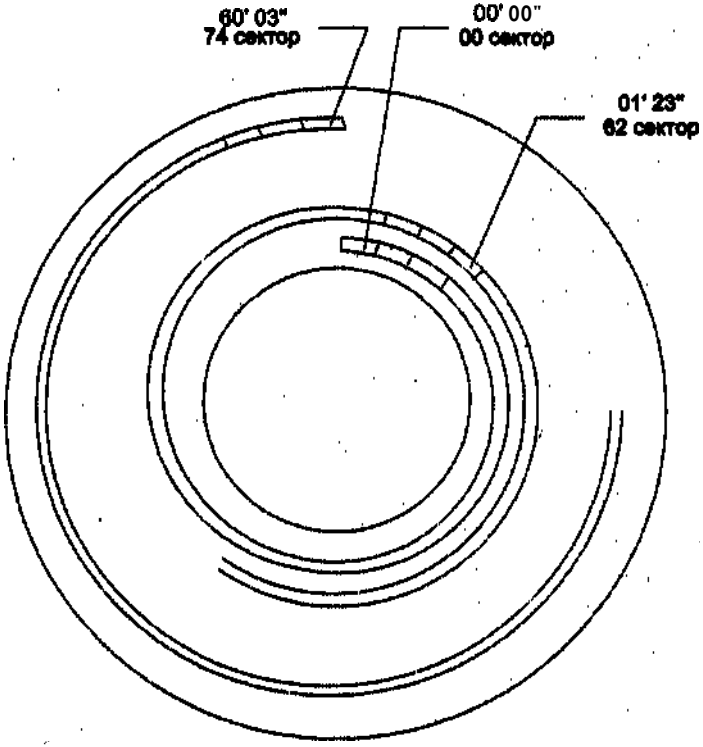


Рис. 5.49. Организация диска с постоянной линейной скоростью

Рисунок 5.49 иллюстрирует организацию информации на CD-ROM. Как уже отмечалось, данные расположены последовательно по спиралевидной дорожке. Для варианта с постоянной линейной скоростью произвольный доступ к информации становится более сложным.

В последнее время наметился переход к новому типу оптических дисков, так называемым DVD-дискам (Digital Video Data). DVD-диски состоят из двух слоев толщиной 0,6 мм, то есть имеют две рабочих поверхности, и обеспечивают хранение по 4,7 Гбайт на каждой. В DVD-технологии используется лазер с меньшей длиной волны (650 нм против 780 нм для стандартных CD-ROM), а также более изощренная схема коррекции. Все это позволяет увеличить число дорожек и повысить плотность записи. Кроме того, при записи применяется метод сжатия информации, известный как MPEG2.

WORM

Технология дисков WORM - дисков с однократной записью и многократным считыванием, была разработана для мелкосерийного производства оптических дисков. Такие диски предполагают ввод информации лучом относительно мощного лазера. При этом пользователь с помощью несколько более дорогого, чем CD-ROM, устройства может единожды записать информацию, а затем многократно ее считывать. Для обеспечения более быстрого доступа в устройстве поддерживается метод постоянной угловой скорости при относительном снижении емкости.

Типовая техника подготовки такого диска предполагает мощный лазер для создания на поверхности диска последовательности пузырьков. Для записи информации предварительно отформатированный пузырьками диск помещается в накопитель WORM, где имеется маломощной лазер, тепла от которого тем не менее достаточно для того, чтобы «взорвать» пузырек. В процессе операции считывания лазер в накопителе WORM освещает поверхность диска. Так как «взорванный» пузырек создает более высокий контраст, чем окружающая поверхность, его легко распознать с помощью простой электроники. Данный тип носителя привлекателен для архивного хранения документов и файлов.

EOD - оптические диски со стиранием

Среди многих рассматривавшихся технологий оптических дисков с возможностью многократной записи и перезаписи информации коммерчески приемлемой оказалась только магнитооптическая. В таких системах энергия лазерного луча используется совместно с магнитным полем. Запись и стирание информации происходят за счет реверсирования магнитных полюсов маленьких областей диска, покрытого магнитным материалом. Лазерный луч нагревает облучаемое пятно на поверхности, и в этот момент магнитное поле может изменить ориентацию магнитных полюсов на облучаемом участке. Поскольку процесс поляризации не вызывает физических изменений на диске, ему не страшны многократные повторения. При чтении направление магнитного поля можно определить по поляризации лазерного луча. Поляризованный свет, отраженный от определенного пятна, изменяет свой угол отражения в зависимости от характера намагниченности.

Магнитные ленты

ЗУ на базе магнитных лент используются в основном для архивирования информации. Носителем служит тонкая полистироловая лента шириной от 0,38-2,54 см и толщиной около 0,025 мм, покрытая магнитным слоем. Лента наматывается на бобины различного диаметра. Данные записываются последовательно, байт за байтом, от начала ленты до ее конца. Время доступа к информации на магнитной ленте значительно больше, чем у ранее рассмотренных видов внешней памяти.

Обычно вдоль ленты располагается 9 дорожек, что позволяет записывать поперек ленты байт данных и бит паритета. Информация на ленте группируется в блоки — *записи*. Каждая запись отделяется от соседней *межблочным промежутком,* дающим возможность позиционирования головки считывания/записи на начало любого блока. Идентификация записи производится по полю заголовка, содержащемуся в каждой записи. Для указания начала и конца ленты используются физические маркеры в виде металлизированных полосок, наклеиваемых на магнитную ленту, или прозрачных участков на самой ленте. Известны также варианты маркирования начала и конца ленты путем записи на нее специальных кодов-индикаторов.

В универсальных ВМ обычно применяются бобинные устройства с вакуумными системами стабилизации скорости перемещения ленты. В них скорость перемещения ленты составляет около 300 см/с, плотность записи — 4 Кбайт/см, а скорость передачи информации - 320 Кбайт/с. Типовая бобина содержит 730 м магнитной ленты.

В ЗУ на базе картриджей используются кассеты с двумя катушками, аналогичные стандартным аудиокассетам. Типовая ширина ленты — 8 мм. Наиболее распространенной формой таких ЗУ является DAT (Digital Audio Tape). Данные на ленту заносятся по диагонали, как это принято в видеокассетах. По размеру такой картридж примерно вдвое меньше, чем обычная компакт-кассета, и имеет толщину 3,81 мм. Каждый картридж позволяет хранить несколько гигабайтов данных. Время доступа к данным невелико (среднее между временами доступа к дискетам и к жестким дискам). Скорость передачи информации выше, чему дискет, но ниже, чем у жестких дисков.

Вторым видом ЗУ на базе картриджей является устройство стандарта DDS (Digital Data Storage). Этот стандарт был разработан в 1989 году для удовлетворения требований к резервному копированию информации с жестких дисков в мощных серверах и многопользовательских системах. В сущности, это вариант DAT, обеспечивающий хранение 2 Гбайт данных при длине ленты 90 м. В более позднем варианте стандарта DDS-DC (Digital Data Storage - Data Compression) за счет применения методов сжатия информации емкость ленты увеличена до 8 Гбайт. Наконец, третий вид ЗУ на базе картриджей также предназначен для резервного копирования содержимого жестких дисков, но при меньших объемах такой информации. Этот тип ЗУ отвечает стандарту QIC (Quarter Inch Cartridge tape) и более известен под названием *стример.* Известны стримеры, обеспечивающие хранение от 15 до 525 Мбайт информации. В зависимости от информационной емкости и фирмы-изготовителя изменяются и характеристики таких картриджей. Так, число дорожек может варьироваться в диапазоне от 4 до 28, длина ленты — от 36 до 300 м и т. д.

Контрольные вопросы

1. Какие операции определяет понятие «обращение к ЗУ»?
2. Какие единицы измерения используются для указания емкости запоминающих устройств?
3. В чем отличие между временем доступа и периодом обращения к запоминающему устройству?
4. Чем вызвана необходимость построения системы памяти по иерархическому принципу?
5. Что включает в себя понятие «локальность по обращению»?
6. Благодаря чему среднее время доступа в иерархической системе памяти определяется более быстродействующими видами ЗУ?
7. Что в иерархической системе памяти определяют термины «промах» и «попадание»?
8. На какие вопросы необходимо ответить, чтобы охарактеризовать определенный уровень иерархической памяти?
9. Какие виды запоминающих устройств может содержать основная память?
10. Охарактеризуйте возможные варианты построения блочной памяти.
11. Какие возможности по сокращению времени доступа к информации предоставляет блочная организация памяти?
12. Чем обусловлена эффективность расслоения памяти?
13. Какая топология запоминающих элементов лежит в основе организации полупроводниковых ЗУ?
14. Какое минимальное количество линий должен содержать столбец ИМС памяти?
15. Поясните назначение управляющих сигналов в микросхеме памяти.
16. Чем отличаются страничный, быстрый страничный и пакетный режимы доступа к памяти?
17. Чем обусловлена необходимость регенерации содержимого динамических ОЗУ?
18. Охарактеризуйте основные сферы применения статических и динамических ОЗУ.
19. Какое влияние на асинхронный режим работы памяти оказывает синхронный характер работы контроллера памяти?
20. В чем состоит особенность подхода, применяемого в микросхемах ОЗУ типа RDRAM и SLDRAM?
21. Какой вид ПЗУ обладает наиболее высокой скоростью перепрограммирования?
22. Какими методами обеспечивается энергонезависимость ОЗУ?
23. В чем состоит различие между режимами стандартной и запаздывающей записи в статических ОЗУ?
24. В чем проявляется специфика ОЗУ, предназначенных для видеосистем?

25. Каким образом в многопортовых ОЗУ разрешаются конфликты при одновременном доступе к памяти?
26. Какую функцию выполняет система семафоров в многопортовой памяти?
27. Для каких целей предназначена память типа FIFO?
28. Какая идея лежит в основе систем обнаружения и коррекции ошибок?
29. Какие ошибки может обнаруживать схема контроля по паритету?
30. От чего зависят возможности выявления и коррекции ошибок с использованием кода Хэмминга?
31. Поясните назначение и принцип формирования кода синдрома в системе коррекции ошибок.
32. Чем объясняется тенденция размещения стека в области старших адресов основной памяти?
33. Какая информация хранится в указателе стека?
34. Поясните назначение маски в ассоциативном запоминающем ЗУ.
35. Как реализуется запись новой информации в ассоциативное ЗУ?
36. Какие виды поиска можно осуществлять в ассоциативном ЗУ?
37. Поясните назначение и логику работы кэш-памяти.
38. Какие проблемы порождает включение в иерархию ЗУ кэш-памяти?
39. Чем обусловлено разнообразие методов отображения основной памяти на кэш-память?
40. Какому требованию должен отвечать «идеальный» алгоритм замещения содержимого кэш-памяти?
41. Какими методами обеспечивается согласованность содержимого основной и кэш-памяти?
42. Чем обусловлено введение дополнительных уровней кэш-памяти?
43. Какие факторы влияют на выбор емкости кэш-памяти и размера блока?
44. Как соотносятся характеристики обычной и дисковой кэш-памяти?
45. Какими средствами обеспечивается виртуализация памяти?
46. Существует ли ограничение на размер виртуального пространства?
47. Что определяет объем страничной таблицы?
48. Какими приемами достигают сокращения объема страничных таблиц?
49. Какие алгоритмы замещения используются при загрузке в основную память новой виртуальной страницы?
50. Поясните назначение буфера быстрого преобразования адреса (TLB).
51. Чем мотивируется разбиение виртуальных секторов на страницы?
52. Какая часть виртуального адреса остается неизменной при его преобразовании в физический адрес?
53. Чем обусловлена необходимость защиты памяти?

Глава 6
Устройства управления

Данная глава освещает различные аспекты структурной организации и функционирования устройства управления вычислительной машины.

Функции центрального устройства управления

Устройство управления (УУ) вычислительной машины реализует функции управления ходом вычислительного процесса, обеспечивая автоматическое выполнение команд программы. Процесс выполнения программы в ВМ представляет собой последовательность машинных циклов. Детализируем основные целевые функции, реализуемые устройством управления в ходе типового машинного цикла [25]. Для простоты примем, что ВМ обеспечивает одноадресную систему команд. При этом, в частности, полагается, что до начала выполнения двухоперандной арифметической команды второй операнд уже находится в процессоре.

Первым этапом в машинном цикле является *выборка команды* из памяти (этап ВК). Целевую функцию этого этапа будем обозначать как ЦФ-ВК.

За выборкой команды следует этап декодирования ее операционной части (кода операции). Для простоты пока будем рассматривать этот этап в качестве составной части этапа ВК.

Вторая целевая функция - *формирование адреса следующей команды*. На это выделяется специальный такт работы — этап ФАСК, которому соответствует целевая функция ЦФ-ФАСК.

Далее следует этап *формирования исполнительного адреса операнда* или адреса перехода (этап ФИА), на котором УУ реализует функцию ЦФ-ФИА. Функция имеет столько модификаций, сколько способов адресации (СА) предусмотрено в системе команд ВМ.

На четвертом этапе реализуется целевая функция *выборки операнда* (ЦФ-ВО) из памяти по исполнительному адресу, сформированному на предыдущем этапе.

Наконец, на последнем этапе машинного цикла действия задаются целевой функцией *исполнения операции* - ЦФ-ИО. Очевидно, что количество модификаций ЦФ-ИО равно количеству операций, имеющихся в системе команд ВМ.

Порядок следования целевых функций полностью определяет динамику работы устройства управления и всей ВМ в целом. Этот порядок удобно задавать и отображать в виде *граф-схемы этапов* исполнения команды (ГСЭ). Как и граф-схема микропрограммы, ГСЭ содержит начальную, конечную, операторные и условные вершины. В начальной и конечной вершинах проставляется условное обозначение конкретной команды, а в условной вершине записывается логическое условие, влияющее на порядок следования этапов. В операторные вершины вписываются операторы этапов.

По форме записи оператор этапа — это оператор присваивания, в котором:

- слева от знака присваивания указывается наименование результата действий, выполненных на этапе;
- справа от знака присваивания записывается идентификатор целевой функции, определяющей текущие действия, а за ним (в скобках) приводится список исходных данных этапа.

Исходной информацией для первого этапа служит хранящийся в счетчике команд адрес A_{Ki} текущей команды Ki. Процесс выборки команды отображается оператором первого этапа: $K := ВК(A_{Ki})$.

Адрес A_{Ki} обеспечивает также второй этап, результатом которого является адрес следующей команды A_{Ki+1}, поэтому оператор второго этапа имеет вид: $A_{Ki+1} :=$ ФАСК (A_{Ki}).

В качестве исходных данных для третьего этапа машинного цикла выступают содержащиеся в коде текущей команды способ адресации CA_i (он определяет конкретную модификацию ЦФ-ФИАО) и код адресной части A_i. Результатом становится исполнительный адрес операнда $A_{исп} :=$ ФИА(C_{Ai}, Ai).

Полученный адрес используется на четвертом этапе для выборки операнда $0_i := ВО(A_{исп})$.

Результат исполнения операции $P0_i$, получаемый на пятом этапе машинного цикла, зависит от кода операции *i*-й команды $КОп_i$ (определяет модификацию ЦФ-ИО), кода первого операнда **0** и кода второго операнда — результата предыдущей $(i-1)$-й операции $P0_{i-1}$: $P0_i :=$ ИО$(КОп_i, 0_i, P0_{i-1})$.

В соответствии со структурой граф-схемы этапов все команды **ВМ** можно разделить на три типа:

- команды типа «Сложение» (Сл);
- команды типа «Запись» (Зп);
- команды типа «Условный переход» (УП).

Типовые граф-схемы этапов представлены на рис. 6.1.

Видно, что количество этапов в командах типа «Сл» (см. рис. 6.1, **а)** колеблется от трех (для непосредственной адресации НА) до пяти. При непосредственной адресации второй операнд записан в адресной части команды, поэтому нет необходимости в реализации устройством управления целевых функций **ЦФ-ФИА,** ЦФ-ВО. Количество этапов для команд типа «Зп» постоянно и равно четырем (см. рис. 6.1, *б)* — здесь отсутствует необходимость в ЦФ-ВО. Машинный цикл команд типа «УП» состоит из трех этапов (см. **рис.** 6.1, *в),* поскольку здесь, помимо выборки операнда, можно исключить и этап ФАСК — действия, обычно выполняемые на этом этапе, фактически реализуются на этапе ИО.

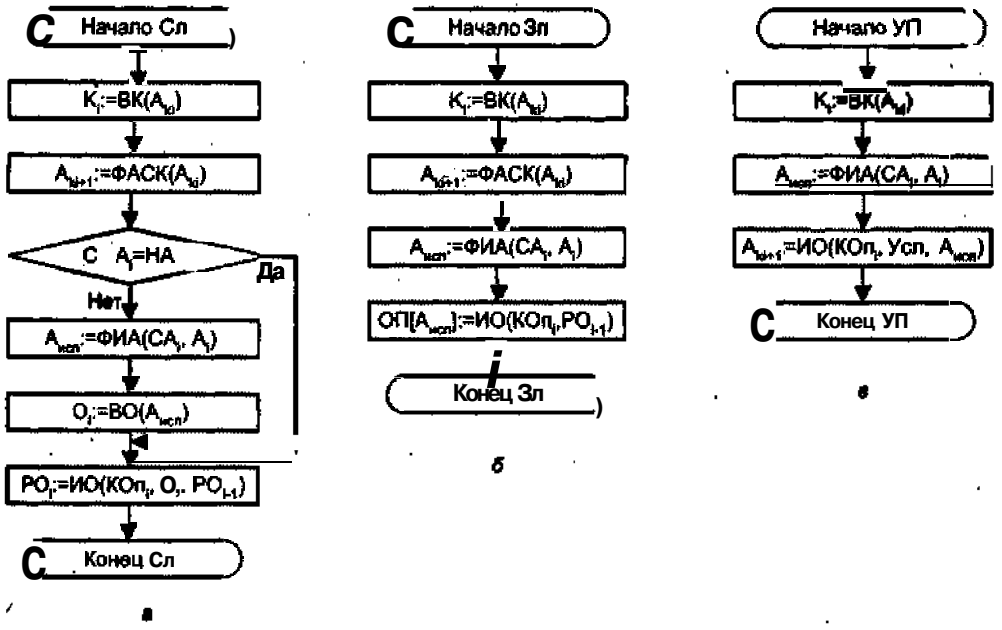


Рис. 6.1. Типовые граф-схемы этапов исполнения команды: *а* - «Сложение»; *б* - «Запись»; *в* — «Условный переход»

Оператор этапа исполнения операции для команд "Зп" имеет смысл записи результата предыдущей операции $РО_{i-1}$ в ячейку с адресом $A_{исп}$:

$$ОП[A_{исп}] := ИО(КОп_i, РО_{i-1}).$$

Местоположение $РО_{i-1}$ определяется кодом операции $КОп_i$. Оператор этапа ИО для команд «**УП**» обеспечивает формирование адреса следующей *(i* + 1)-й команды в зависимости от $A_{исп}$ и значения проверяемого условия перехода Усл

$$A_{ki+1} := ИО(КОп_i, Усл, A_{исп}).$$

Местоположение проверяемого условия также определяется кодом операции КОп

Модель устройства управления

Для выполнения своих функций УУ должно иметь входы, позволяющие определить состояние управляемой системы, и выходы, через которые реализуется управление поведением системы. Модель УУ показана на рис. 6.2 [200] .

Входной информацией для устройства управления служат:

- *тактовые импульсы* - с каждым тактовым импульсом УУ инициирует выполнение одной или нескольких микроопераций;
- *код операции* — код операции текущей команды поступает из регистра команды и используется, чтобы определить, какие микрооперации должны выполняться в течение машинного цикла;

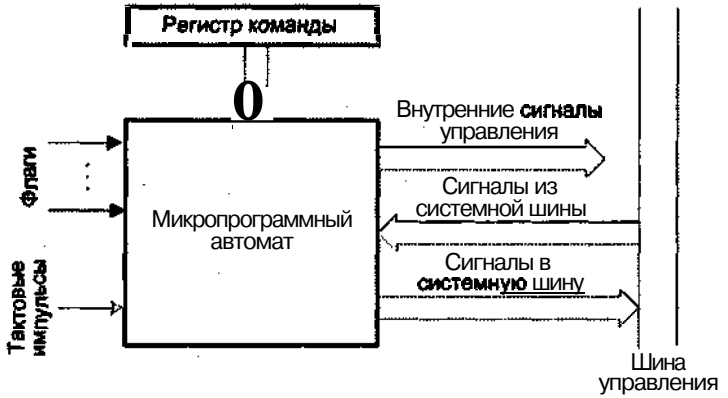


Рис. 6.2. Модель устройства управления

- *флаги* — требуются устройству управления для оценки состояния ЦП и результата предшествующей операции, что необходимо при выполнении команд условного перехода;
- *сигналы из системной шины* — часть сигналов с системной шины, обеспечивающая передачу в УУ запросов прерывания, подтверждений и т. п.

В свою очередь, УУ, а точнее микропрограммный автомат, формирует следующую выходную информацию:

- *внутренние сигналы управления* — эти сигналы воздействуют на внутренние схемы центрального процессора и относятся к одному из двух типов: тем, которые вызывают перемещение данных из регистра в регистр, и тем, что инициируют определенные функции операционного устройства ВМ;
- *сигналы в системную шину* — также относятся к одному из двух типов: управляющие сигналы в память и управляющие сигналы в модули ввода/вывода.

Структура устройства управления

Как уже отмечалось ранее, процесс функционирования **ВМ** состоит из последовательности элементарных действий в ее узлах. Такие элементарные преобразования информации, выполняемые в течение одного такта сигналов синхронизации, *называются микрооперациями* (МО). Совокупность сигналов управления, вызывающих одновременно выполняемые микрооперации, образует *микрокоманду* (МК). В свою очередь, последовательность микрокоманд, определяющую содержание и порядок реализации машинного цикла, принято называть *микропрограммой*. Сигналы управления вырабатываются устройством управления, а точнее одним из его узлов - *микропрограммным автоматом* (МПА). Название отражает то, что МПА определяет микропрограмму как последовательность выполнения микроопераций.

Микропрограммы реализации перечисленных ранее целевых функций инициируются *задающим оборудованием,* которое вырабатывает требуемую последовательность сигналов управления и входит в состав управляющей части УУ.

Выполняются микропрограммы *исполнительным оборудованием,* входящим в состав основной памяти (для ЦФ-ВК и ЦФ-ВО) и операционного устройства (для ЦФ-ИО). Исполнительным оборудованием для целевых функций ЦФ-ФАСК, ЦФ-ФИА служит адресная часть устройства управления. В обобщенной структуре УУ (рис. 6.3) можно выделить две части: управляющую и адресную.

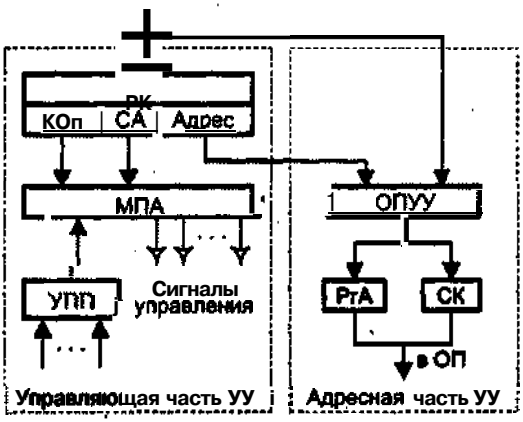


Рис. 6.3. Обобщенная структура устройства управления

Управляющая часть УУ предназначена для координирования работы операционного блока ВМ, адресной части устройства управления, основной памяти и других узлов ВМ.

Адресная часть УУ обеспечивает формирование адресов команд и исполнительных адресов операндов в основной памяти.

В состав управляющей части УУ входят:

- регистр команды (РК), состоящий из адресной (Адрес) и операционной (КОп, СА) частей;
- микропрограммный автомат (МПА);
- узел прерываний иприоритетов УПП).

Регистр команды РК предназначен для приема очередной команды из запоминающего устройства. Микропрограммный автомат на основании результатов расшифровки операционной части команды (КОп, СА) вырабатывает определенную последовательность микрокоманд, вызывающих выполнение всех целевых функций УУ.

В зависимости от способа формирования микрокоманд различают *микропрограммные автоматы:*

- с жесткой или аппаратной логикой;
- с программируемой логикой.

Организация МПА этих двух типов будет рассмотрена в последующих разделах.

Узел прерываний и приоритетов позволяет реагировать на различные ситуации, связанные как с выполнением рабочих программ, так и с состоянием ВМ.

Адресная часть УУ включает в себя:

- операционный узел устройства управления (ОПУУ);
- регистр адреса (РгА);
- счетчик команд (СК).

Регистр адреса используется для хранения исполнительных адресов операндов, а *счетчик команд* - для выработки и хранения адресов команд. Содержимое РгА и СК посылается в регистр адреса основной памяти (ОП) для выборки операндов и команд соответственно.

ОПУУ, называемый иначе узлом индексной арифметики или узлом адресной арифметики, обрабатывает адресные части команд, формируя исполнительные адреса операндов, а также подготавливает адрес следующей команды при выполнении команд перехода. Состав ОПУУ может быть аналогичен составу основного операционного устройства ВМ (иногда в простейших ВМ с целью экономии затрат на оборудование ОПУУ совмещается с основным операционным устройством).

Сказанное об адресной части УУ проиллюстрируем примерами. Пусть в ОПУУ входят два индексных регистра $ИР_1$ $ИР_2$ и индексный сумматор СМИ, как показано на рис. 6.4.

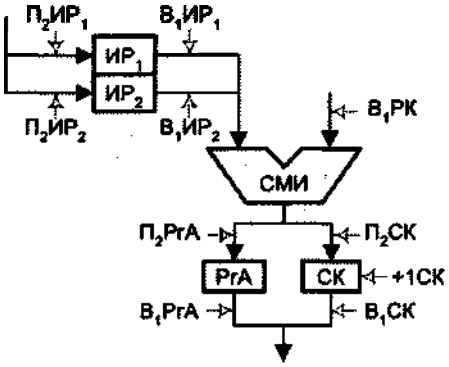


Рис. 6.4. Пример реализации адресной части УУ

Для данной схемы микропрограмма формирования исполнительного адреса имеет вид, представленный на рис. 6.5, *а.*

ПРИМЕЧАНИЕ Индексы при сокращениях П (Прием) и В (Выдача) обозначают фазность передаваемого кода. Каждый двоичный разряд однофазного кода передается по одной цепи (и поступает только на вход S соответствующего триггера). Каждый двоичный разряд парафазного кода передается по двум цепям (и поступает на входы S и R соответствующего триггера), при этом не требуется предварительное обнуление триггера-приемника.

Выполняемые действия определяются полем способа адресации. Если СА указывает на индексную адресацию относительно $ИР_1$ или $ИР_2$ (СА = 1, СА = 2), то по

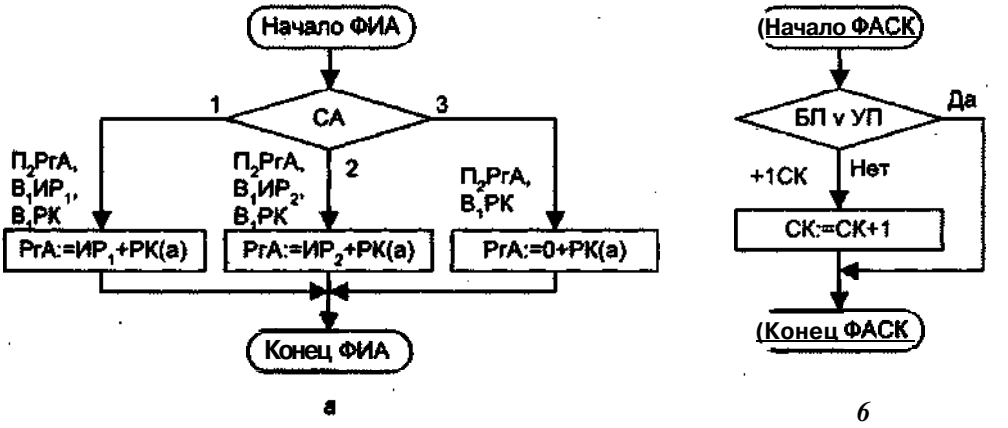


Рис. 6.5. Микропрограмма формирования: а - исполнительного адреса; *б* -- адреса следующей команды

управляющему сигналу **В,ИР** ($В_1ИР_2$) индекс адреса из **ИР** ($ИР_2$) подается на левый вход сумматора **СМИ.** Одновременно по управляющему сигналу ВР К на правый вход С М*И* поступает адресная часть команды из регистра команды - РК(а). Осуществляется микрооперация сложения, результат которой ($А_{исп}$) по управляющему сигналу $П_2$**РгА** заносится в РгА. Если СА = 3, то адрес формируется по способу прямой адресации. В этом случае по управляющему сигналу **В,РК** выполняется микрооперация сложения адресной части РК с нулем. Результат сложения по управляющему сигналу $П_2$**РгА** с выхода СМИ записывается в РгА

Микропрограмма формирования адреса следующей команды (ЦФ-ФАСК) изображена на рис. 6.5, *б.* Видим, что естественное формирование адреса следующей команды (с помощью СК) не производится, если исполняется команда безусловного (БП) или условного (УП) перехода. Такой адрес формируется на этапах ФИА и ИО, он равен исполнительному адресу (если это УП и условие перехода выполняется, или если это БП).

В состав УУ могут также входить дополнительные узлы, в частности узел организации прямого доступа к памяти. Этот узел обычно реализуется в виде самостоятельного устройства — контроллера прямого доступа к памяти (КПДП). КПДП обеспечивает совмещение во времени работы операционного устройства с процессом обмена информацией между **ОП** и другими устройствами ВМ, тем самым повышая общую производительность машины.

Довольно часто регистры различных узлов УУ объединяют в отдельный узел управляющих (специальных) регистров устройства управления.

Все множество технологий, используемых при реализации микропрограммных автоматов устройств управления, можно свести к двум категориям:

- МПА с «жесткой» логикой или аппаратурной реализацией;
- МПА с программируемой логикой.

Микропрограммный автомат с жесткой логикой

Обычно тип микропрограммного автомата (МПА), формирующего сигналы управления, определяет название всего УУ. Так, УУ с жесткой логикой управления имеет в своем составе МПАс жесткой (аппаратной) логикой. При создании такого МПА выходные сигналы управления реализуются за счет однажды соединенных между собой логических схем.

Типичная структура микропрограммного автомата с жесткой логикой управления показана на рис. 6.6.

Исходной информацией для УУ (см. рис. 6.2) служат: содержимое регистра команды, флаги, тактовые импульсы и сигналы, поступающие с шины управления.

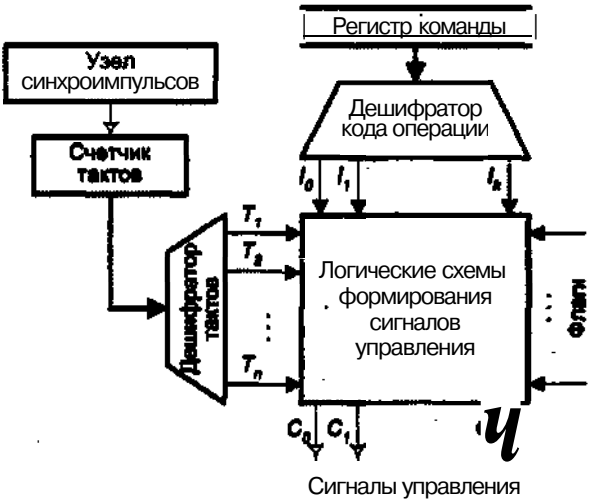


Рис. 6.6. Микропрограммный автомат с жесткой логикой

Код операции, хранящийся в РК, используется для определения того, какие СУ и в какой последовательности должны формироваться, при этом, с целью упрощения логики управления, желательно иметь в УУ отдельный логический сигнал для каждого кода операции $(I_0, I_1, ..., I_k)$.Это может быть реализовано с помощью дешифратора. Дешифратор кода операции преобразует код *j*-й операции, поступающей из регистра команды (РК), в единичный сигнал на *j*-м выходе.

Машинный цикл выполнения любой команды состоит из нескольких тактов. Сигналы управления, по которым выполняется каждая микрооперация, должны вырабатываться в строго определенные моменты времени, поэтому все СУ «привязаны» к импульсам синхронизации (СИ), формируемым узлом синхроимпульсов. Период СИ должен быть достаточным для того, чтобы сигналы успели распространиться по трактам Данных и другим цепям. Каждый СУ ассоциируется с одним из тактовых периодов в рамках машинного цикла. Формирование сигналов, отмечающих начало очередного тактового периода, возлагается на синхрониза-

тор. Синхронизатор содержит счетчик тактов, осуществляющий подсчет СИ. Узел синхроимпульсов после завершения очередного такта работы добавляет к содержимому счетчика тактов единицу. К выходам счетчика подключен дешифратор тактов, с которого и снимаются сигналы тактовых периодов: T_1..., Tn. В i-м состоянии счетчика тактов, то есть во время i-го такта, дешифратор тактов вырабатывает единичный сигнал на своем *i-м* выходе. При такой организации в УУ должна быть предусмотрена обратная связь, с помощью которой по окончании цикла команды счетчик тактов опять устанавливается в состояние $Г_1$.

Дополнительным фактором, влияющим на последовательность формирования СУ, являются состояние осведомительных сигналов (флагов), отражающих ход вычислений, и сигналы с шины управления. Эта информация также поступает на вход УУ, причем каждая линия здесь рассматривается независимо от остальных.

Процесс синтеза схемы МПА с жесткой логикой называется структурным синтезом и разделяется на следующие этапы:

- выбор типа логических и запоминающих элементов;
- кодирование состояний автомата;
- синтез комбинационной схемы, формирующей выходные сигналы.

Чтобы определить способ реализации МПА с жесткой логикой, необходимо описать внутреннюю логику УУ, формирующую выходные сигналы управления, как булеву функцию входных сигналов. Канонический метод структурного синте-

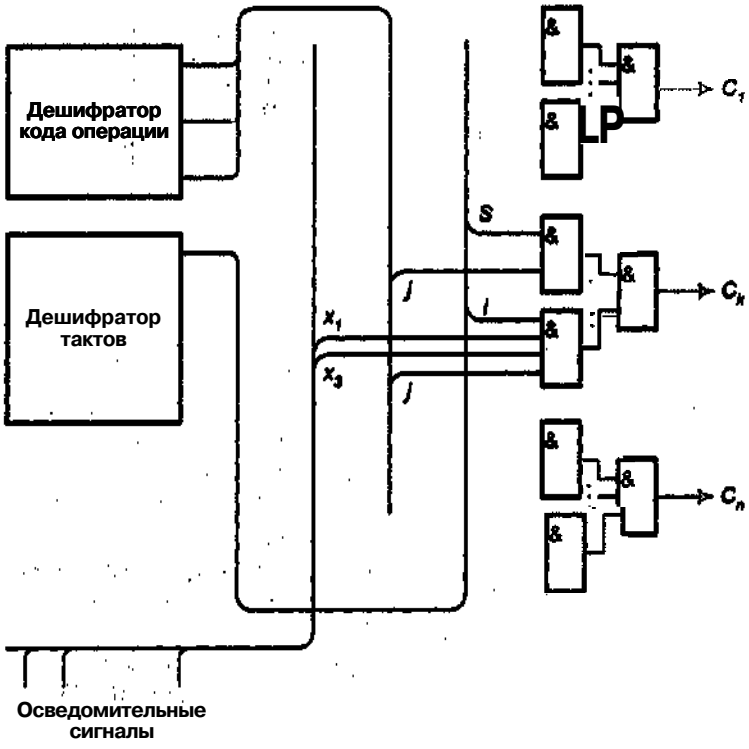


Рис. в.7. Фрагмент схемы формирования сигналов управления

за МПА был предложен В. М. Глушковым. Согласно этому методу, задача синтеза автомата сводится к синтезу комбинационной схемы путем построения системы логических функций с последующей их минимизацией.

Принцип построения логических схем формирования управляющих сигналов поясняется на рис. 6.7. Здесь показан фрагмент схемы, обеспечивающей выработку управляющего сигнала C_k в *i-м* и *S-м* тактах выполнения команды с кодом операции *j*, причем сигнал C_k появляется в *i-м* такте только при значениях осведомительных сигналов x1=1 и x3=1 S-м такте всегда.

Таким образом, название «жесткая логика» обусловлено тем, что каждой микропрограмме здесь соответствует свой набор логических схем с фиксированными связями между ними. При реализации простой системы команд узлы МПА с жесткой логикой экономичны и позволяют обеспечить наибольшее быстродействие из всех возможных методов построения МПА. Однако с возрастанием сложности системы команд соответственно усложняются и схемы автоматов с жесткой логикой, в результате чего уменьшается их быстродействие. Второй недостаток МПА с жесткой логикой — малая регулярность, а следовательно, и большие трудности при размещении УУ такого типа на кристалле интегральной микросхемы.

Микропрограммный автомат с программируемой логикой

Принципиально иной подход, позволяющий преодолеть сложность УУ с жесткой логикой, был предложен британским ученым М. Уилксом в начале 50-х годов [224][1]. В основе идеи лежит тот факт, что для инициирования любой микрооперации достаточно сформировать соответствующий СУ на соответствующей линии управления, то есть перевести такую линию в активное состояние. Это может быть представлено с помощью двоичных цифр 1 (активное состояние — есть СУ) и О (пассивное состояние - нет СУ). Для указания микроопераций, выполняемых в данном такте, можно сформировать управляющее слово, в котором каждый бит соответствует одной управляющей линии. Такое управляющее слово называют *микрокомандой* (МК). Таким образом, микрокоманда может быть представлена управляющим словом со своей комбинацией нулей и единиц. Последовательность микрокоманд, реализующих определенный этап машинного цикла, образует *микропрограмму*. В терминологии на английском языке микропрограмму часто называют *firmware,* подчеркивая тот факт, что это нечто среднее между аппаратурой *(hardware)* и программным обеспечением *(software)*. Микропрограммы для каждой команды ВМ и для каждого этапа цикла команды размещаются в специальном ЗУ, называемом *памятью микропрограмм* (ПМК). Процесс формирования СУ можно реализовать, последовательно (с каждым тактовым импульсом) извлекая микрокоманды микропрограммы из памяти и интерпретируя содержащуюся в них информацию о сигналах управления.

[1] Аналогичную идею, независимо от Уилкса, в 1957 году выдвинул российский ученый Н. Я. Матюхин. Предложенное им УУ с программируемой логикой было реализовано в 1962 году в специализированной ВМ «Тетива», предназначенной для системы противовоздушной обороны.

Идея заинтересовала многих конструкторов ВМ, но была нереализуема, поскольку требовала использования быстрой памяти относительно большой емкости. Вновь вернулись к ней в 1964 году, в ходе создания системы IBM 360. С тех пор устройства управления с программируемой логикой стали чрезвычайно популярными и были встроены во многие ВМ. В этой связи следует упомянуть запатентованный академиком В. М. Глушковым принцип ступенчатого микропрограммирования, который он впервые реализовал в машине «Промінь».

Принцип управления по хранимой в памяти микропрограмме

Отличительной особенностью микропрограммного автомата с программируемой логикой является хранение микрокоманд в виде кодов в специализированном запоминающем устройстве - памяти микропрограмм. Каждой команде ВМ в этом ЗУ в явной форме соответствует микропрограмма, поэтому часто устройства управления, в состав которых входит микропрограммный автомат с программируемой логикой, называют микропрограммными.

Типичная структура микропрограммного автомата представлена на рис. 6.8. В составе узла присутствуют: память микропрограмм (ПМП), регистр адреса микрокоманды (РАМ), регистр микрокоманды (РМК), дешифратор микрокоманд (ДшМК), преобразователь кода операции, формирователь адреса следующей микрокоманды (ФАСМ), формирователь синхроимпульсов (ФСИ).

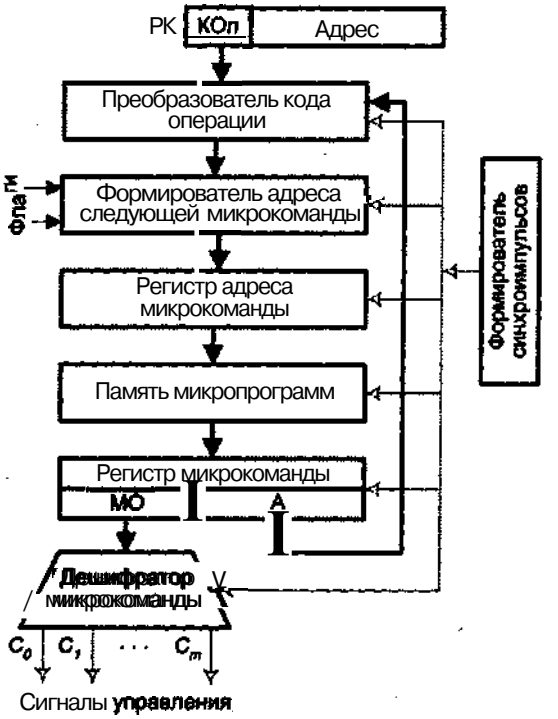


Рис. 6.8. Микропрограммный автомат с программируемой логикой

Запуск микропрограммы выполнения операции осуществляется *путем* передачи кода операции из РК на вход преобразователя, в котором код операции преобразуется в начальный (первый) адрес микропрограммы $A_{н}$. Этот адрес поступает через ФАСМ в регистр адреса микрокоманды. Выбранная по адресу $A_{н}$ из ПМП микрокоманда заносится в РМК. Каждая микрокоманда в общем случае содержит микрооперационную (МО) и адресную (А) части. **Микрооперационная** часть микрокоманды поступает на дешифратор микрокоманды, на выходе которого образуются управляющие сигналы, инициирующие выполнение микроопераций в исполнительных устройствах и узлах ВМ. Адресная часть микрокоманды подается в ФАСМ, где формируется адрес следующей микрокоманды $A_{мк}$. Этотадрес может зависеть от адреса на выходе преобразователя кода операции *Ли,* адресной части текущей микрокоманды *А* и значений осведомительных сигналов (флагов) *X,* поступающих от исполнительных устройств. Сформированный адрес микрокоманды снова записывается в РАМ, и процесс повторяется до окончания микропрограммы.

Разрядность адресной $(R_{А})$ и микрооперационной $(R_{МО})$ частей микрокоманды определяются из соотношений

$$R_{А} = \text{int}(\log_2 N_{МК}); \quad (6.1)$$

$$R_{МО} = \text{int}(\log_2 N_{СУ}). \quad (6,2)$$

где $N_{МК}$ — общее количество микрокоманд; $N_{СУ}$ — общее количество формируемых сигналов управления.

В свою очередь, необходимая емкость памяти микропрограмм равна

$$E_{ПМП} = N_{МК}(R_{А} + R_{МО}) = N_{МК}(\text{int}(\log_2 N_{МК}) + \text{int}(\log_2 N_{СУ})).$$

Кодирование микрокоманд

Информация о том, какие сигналы управления должны быть сформированы в процессе выполнения текущей МК, в закодированном виде содержится в микрооперационной части (МО) микрокоманды. Способ кодирования микроопераций во многом определяет сложность аппаратных средств устройства управления и его скоростные характеристики. Применяемые в микрокомандах варианты кодирования сигналов управления можно свести к трем группам: минимальное кодирование (горизонтальное микропрограммирование), максимальное кодирование (вертикальное микропрограммирование) и групповое кодирование (смешанное микропрограммирование). Структуры микропрограммных автоматов при различных способах кодирования микроопераций показаны на рис. 6.9 **[12, 28].**

При горизонтальном микропрограммировании (см. рис. 6.9, *а)* под каждый сигнал управления в микрооперационной части микрокоманды выделен один разряд $(R_{МО} = N_{СУ})$. Это позволяет в рамках одной микрокоманды формировать любые сочетания СУ, чем обеспечивается максимальный параллелизм выполнения микроопераций. Кроме того, отсутствует необходимость в декодировании МО и выходы регистра микрокоманды могут быть непосредственно подключены к соответствующим управляемым точкам ВМ. Широкому распространению горизонтального микропрограммирования тем не менее препятствуют большие затраты на хранение микрооперационных частей микрокоманд $(E_{МО} = N_{МК} * N_{СУ})$, причем эффек-

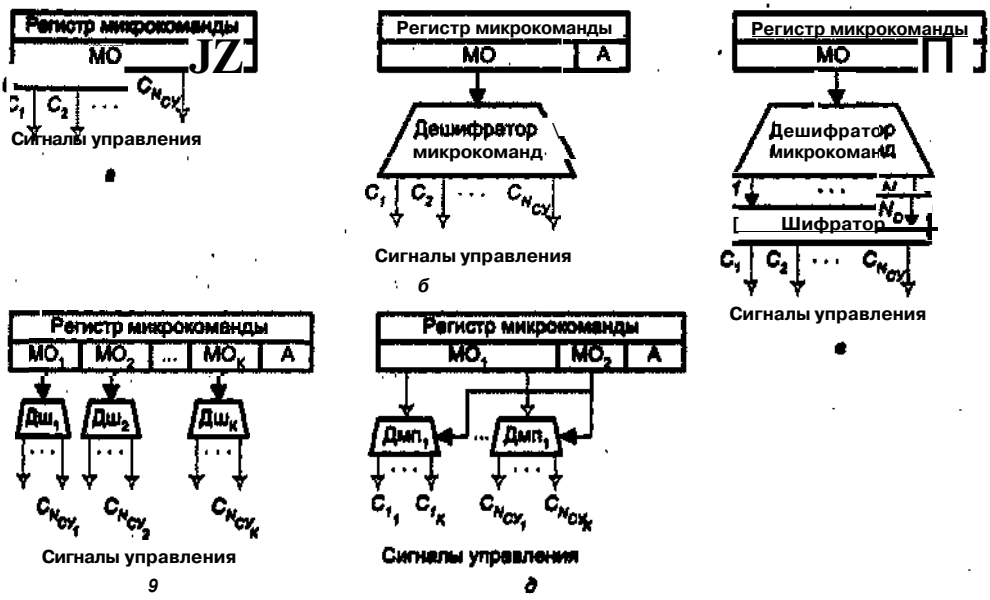


Рис. 6.9. Структуры МПА при различных способах кодирования микроопераций:
а — минимальном; *б* — максимальном; в — максимальном с шифратором;
Г— горизонтально-вертикальном; *д* — вертикально-горизонтальном

тивность использования ПМП получается низкой, так как при большом числе микроопераций в каждой отдельной МК реализуется лишь одна или несколько из них, то есть подавляющая часть разрядов МО содержит нули.

При максимальном (вертикальном) кодировании (см. рис. 6.9, *б)* каждой микрооперации присваивается определенный код, например, ее порядковый номер в полном списке возможных микроопераций. Этот код и заносится в МО.' Микрооперационная часть МК имеет минимальную длину, определяемую как двоичный логарифм от числа управляющих сигналов (микроопераций) по формуле (6.2). Такой способ кодирования требует минимальных аппаратных затрат в ПМП на хранение микрокоманд, однако возникает необходимость в дешифраторе ДшМК, который должен преобразовать код микрооперации в соответствующий сигнал управления. При большом количестве СУ дешифратор вносит значительную временную задержку, а главное - в каждой МК указывается лишь один сигнал управления, инициирующий только одну микрооперацию, за счет чего увеличиваются длина микропрограммы и время ее реализации.

., Последний недостаток устраняется при подключении к выходам ДшМК шифратора (Ш) (см. рис. 6.9, *в*), что приводит к увеличению количества СУ, формируемых одновременно. Естественно, что помимо кодов отдельных микроопераций должны быть предусмотрены коды, представляющие и определенные комбинации микроопераций. Для повышения универсальности шифратор целесообразно строить на базе запоминающего устройства.

Вариант, представленный на рис. 6.9, *в,* рационален, если

$$N_D \ll N_{МК} \text{ и } R_{МО} = \text{int}(\log_2 N_D) < N_{СУ},$$

где N_D – количество выходов дешифратора. При этих условиях аппаратные затраты на хранение микрооперационных частей МК относительно малы:

$$E_{МО} = N_{МК} R_{МО} = N_{МК} \operatorname{int}(\log_2 N_D).$$

Однако полная емкость используемой памяти равна

$$E_{МО} + E_{Ш} = N_{МК} \operatorname{int}(\log_2 N_D) + N_D N_{СУ},$$

откуда ясно, что при близких $N_{МК}$ и N_D вариант теряет смысл.

Минимальное и максимальное кодирование являются двумя крайними точками широкого спектра возможных решений задачи кодирования СУ. Промежуточное положение занимает групповое или смешанное кодирование.

Здесь все сигналы управления (микрооперации) разбиваются на K групп

$$Y = \bigcup_{l=1}^{K} Y_l.$$

В зависимости от принципа разбиения микроопераций на группы различают горизонтально-вертикальное и вертикально-горизонтальное кодирование.

В горизонтально-вертикальном методе (см. рис. 6.9, *г)* в каждую группу включаются взаимно несовместимые сигналы управления (микрооперации), то есть СУ, которые никогда не встречаются вместе в одной микрокоманде. При этом сигналы, обычно формируемые в одном и том же такте, оказываются в разных группах. Внутри каждой группы сигналы управления кодируются максимальным (вертикальным) способом, а группы - минимальным (горизонтальным) способом.

Каждой группе Y_l выделяется отдельное поле в микрооперационной части МК, общая разрядность которой равна

$$R_{МО} = \sum_{l=1}^{K} R_{МО_l} = \sum_{l=1}^{K} \operatorname{int}(\log_2 N_{СУ_l}),$$

где $N_{СУ_l}$ - количество СУ, представляемых l-м полем (группой).

Общая емкость памяти микропрограмм рассмотренного варианта кодирования определяется из выражения:

$$E_{ПМП} = N_{МК}(R_{МО} + R_A) = N_{МК}\left(\sum_{l=1}^{K} \operatorname{int}(\log_2 N_{СУ_l}) + \operatorname{int}(\log_2 N_{МК})\right).$$

При вертикально-горизонтальном способе (см. рис. 6.9, *д)* все множество сигналов управления (микроопераций) также делится на несколько групп, однако в группу включаются сигналы управления (микрооперации), наиболее часто встречающиеся вместе в одном такте. Поле МО делится на две части: МО и $МО_2$. Поле $МО_1$, длина которого равна максимальному количеству сигналов управления (микроопераций) в группе, кодируется горизонтально, а поле $МО_2$, указывающее на принадлежность к определенной группе, — вертикально. Со сменой группы меняются и управляемые точки, куда должны быть направлены сигналы управления из каждой позиции МО. Это достигается с помощью демультиплексоров (Дмп), управляемых кодом номера группы из поля М О ,.

При горизонтальном, вертикальном и горизонтально-вертикальном способах кодирования микроопераций каждое поле микрокоманды несет фиксированные функции, то есть имеет место *прямое кодирование*. При *косвенном кодировании* одно из полей отводится для интерпретации других полей. Примером косвенного кодирования микроопераций может служить вертикально-горизонтальное кодирование.

Иногда используется двухуровневое кодирование микроопераций. На первом уровне с вертикальным кодированием выбирается микрокоманда, поле МО которой является адресом горизонтальной микрокоманды второго уровня — *нанокоманды*. Данный способ сочетания вертикального и горизонтального микропрограммирования часто называют *нанопрограммированием*. Метод предполагает двухуровневую систему кодирования микроопераций и, соответственно, двухуровневую организацию управляющей памяти (рис 6.10).

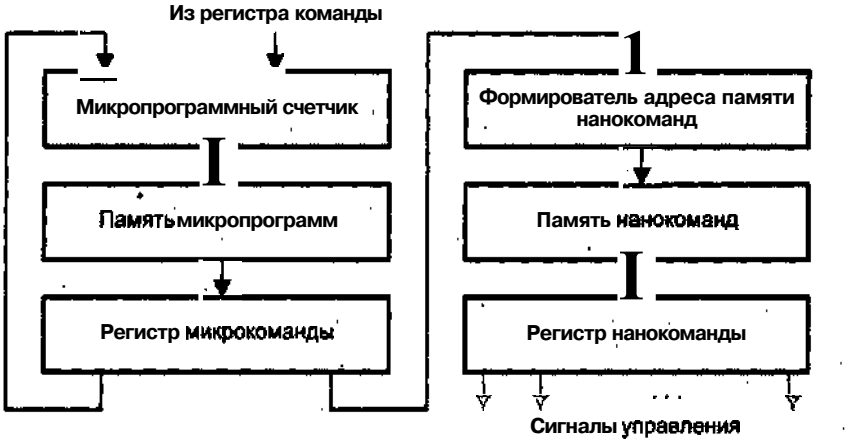


Рис. 6.10. Нанопрограммное устройство управле

Верхний уровень управления образуют микропрограммы, хранящиеся в памяти микропрограмм. В микрокомандах используется вертикальное кодирование микроопераций. Каждой микрокоманде соответствует нанокоманда, хранящаяся в управляющей памяти нижнего уровня — памяти нанокоманд. В нанокомандах применено горизонтальное кодирование микроопераций. Именно нанокоманды используются для непосредственного формирования сигналов управления. Микрокоманды вместо закодированного номера СУ содержат адресную ссылку на соответствующую нанокоманду.

Такой подход, сохраняя все достоинства горизонтального микропрограммирования, позволяет значительно сократить суммарную емкость управляющей памяти. Дело в том, что число различных сочетаний сигналов управления, а следовательно, количество «длинных» нанокоманд, обычно невелико. В то же время практически все микрокоманды многократно повторяются в различных микропрограммах, и замена в микрокомандах «длинного» горизонтального управляющего поля на короткую адресную ссылку дает ощутимый эффект. Проиллюстрируем это примером.

Пусть в системе вырабатывается 200 управляющих сигналов, а общая длина микропрограмм составляет 2048 микрокоманд. Предположим также, что реально используется только 256 различных сочетаний сигналов управления. В случае обычного УУ с горизонтальным микропрограммированием емкость управляющей памяти составила бы 2048 x 200 ,= 409 600 бит. При нанопрограммировании требуется память микропрограмм емкостью 2048 x 8 = 16 384 бит и Память нанокоманд емкостью 256 x 200 = 51 200 бит. Таким образом, суммарная емкость обоих видов управляющей памяти равна 67 584 битам.

Иначе говоря, нанопамять реализует функцию шифратора управляющих сигналов (см. рис. 6.9, *в)*. Справедливы следующие соотношения: $R_{\text{М}} < R_{\text{н}}$, $E_{\text{М}} \gg E_{\text{Н}}$, где E_{M}, $R_{\text{М}}$—емкость и разрядность микропамяти; E_{H} , $R_{\text{Н}}$–емкость и разрядность памяти нанокоманд.

Основным недостатком нанопрограммирования является невысокое быстродействие, поскольку для выполнения микрокоманды требуются два обращения к памяти, что, однако, частично компенсируется исключением из схемы дешифратора, свойственного вертикальному микропрограммированию.

Рис. 6.11. Организация управляющей памяти

Двухуровневая память рассматривается как способ для уменьшения необходимой емкости ПМП, ее использование целесообразно только при многократном повторении микрокоманд в микропрограмме.

При разбиении микрокоманды на поля могут действовать два принципа: по функциональному назначению СУ и по ресурсам. *Функциональное кодирование* предполагает, что каждое поле соответствует одной функции внутри ВМ. Например, информация в аккумулятор может заноситься из различных источников, и для указания источника в микрокоманде может быть отведено одно поле, где каждая кодовая комбинация прикреплена к определенному источнику. При *ресурсном кодировании* ВМ рассматривается как набор независимых ресурсов (память, УВВ, АЛУ и т. п.), и каждому из ресурсов в микрокоманде отводится свое поле.

Обеспечение последовательности выполнения микрокоманд

На рис. 6.11 показано возможное размещение микрокоманд в памяти микропрограмм. Содержимое ПМП определяет последовательность микроопераций, которые должны выполняться на каждом этапе цикла команды, а также последовательность этапов. Каждый этап представлен соответствующей микропрограммой. Микропрограммы завершаются микрокомандой перехода, определяющей последующие действия. В управляющей памяти имеется также специальная микропрограмма-переключатель: в зависимости от текущего кода операции она указывает, этап исполнения какой команды должен быть выполнен.

Большей частью микрокоманды в микропрограмме выполняются последовательно, однако в общем случае очередность микроопераций не является фиксированной. По этой причине в УУ необходимо предусмотреть эффективную систему реализации переходов. Переходы, как безусловные, так и условные, являются неотъемлемой частью любой микропрограммы.

Адресация микрокоманд

При выполнении микропрограммы адрес очередной микрокоманды относится к одной из трех категорий:

- определяется кодом операции команды;
- является следующим по порядку адресом;
- является адресом перехода.

Первый случай имеет место только один раз в каждом цикле команды, сразу же вслед за ее выборкой. Как уже отмечалось ранее, каждой команде из системы команд ВМ соответствует «своя» микропрограмма в памяти микропрограмм, поэтому первое действие, которое нужно произвести после выборки команды, — преобразовать код операции в адрес первой МК соответствующей микропрограммы. Это может быть выполнено с помощью аппаратного преобразователя кода операции (см. рис. 6.8). Такой преобразователь обычно реализуется в виде специального ЗУ, хранящего начальные адреса микропрограмм в ПМП. Для указания того, как должен вычисляться адрес следующей МК в микрокоманде, может быть выделено специальное однобитовое поле. Единица в этом поле означает, что $_{\text{мк}}$ должен быть

сформирован на основании кода операции. Подобная МК обычно располагается в конце микропрограммы этапа выборки или в микропрограмме анализа кода операции.

Дальнейшая очередность выполнения микрокоманд микропрограммы может быть задана путем указания в каждой МК адреса следующей микрокоманды *(принудительная адресация)* либо путем автоматического увеличения на единицу адреса текущей *МК (естественная адресация)* [21].

В обоих случаях необходимо предусмотреть ситуацию, когда адрес следующей микрокоманды зависит от состояния осведомительных сигналов — ситуацию перехода. Для указания того, какое условие должно быть проверено, в МК вводится поле условия перехода (УП). Поле УП определяет номер *i* осведомительного сигнала x_i, значение которого анализируется при формировании адреса следующей микрокоманды. Если поле УП = 0, то никакие условия не проверяются и $_{мк}$ либо берется из адресной части микрокоманды (при принудительной адресации), либо формируется путем прибавления единицы к адресу текущей микрокоманды. Если УП!=0, тогда $_{мк} = A + x_i$. В результате этого осуществляется условный переход: при x_i - 0 к микрокоманде с адресом $_{мк} = A$, а при x_i * 1 — к микрокоманде с адресом $_{мк} = A + 1$. Описанный порядок формирования адреса показан на рис. 6.12 и имеет место в случае принудительной адресации с одним адресом.

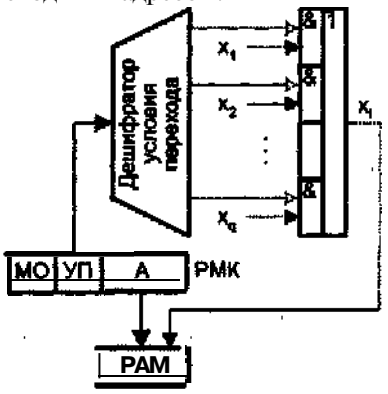


Рис. 6.12. Условные переходы при принудительной адресации микрокоманд

Рассмотренный способ позволяет в каждой команде учесть состояние только одного из осведомительных сигналов. Более гибкий подход реализован в ряде ВМ фирмы IBM. В нем адреса микрокоманды разбиваются на две составляющие. Старшие *n* разрядов обычно остаются неизменными. В процессе выполнения микрокоманды эти разряды просто копируются из адресной части МК в аналогичные позиции РАМ, определяя блок из 2^n микрокоманд в памяти микропрограмм. Остальные (младшие) *k* разрядов РАМ устанавливаются в 1 или 0 в зависимости от того, проверка каких осведомительных сигналов была задана в поле УП и в каком состоянии эти сигналы находятся. Такой метод позволяет в одной микрокоманде сформировать 2^k вариантов перехода.

Теперь рассмотрим возможные способы реализации принудительной и естественной адресации. Известные подходы сводятся к трем типовым вариантам [75]:

- два адресных поля;
- одно адресное поле;
- переменный формат.

Два *первых метода предст*авляют *принудительную адресацию,* а третий - *естественную адресацию* микрокоманд.

Простейшим вариантом является включение в микрокоманду двух адресных полей (рис. 6.13).

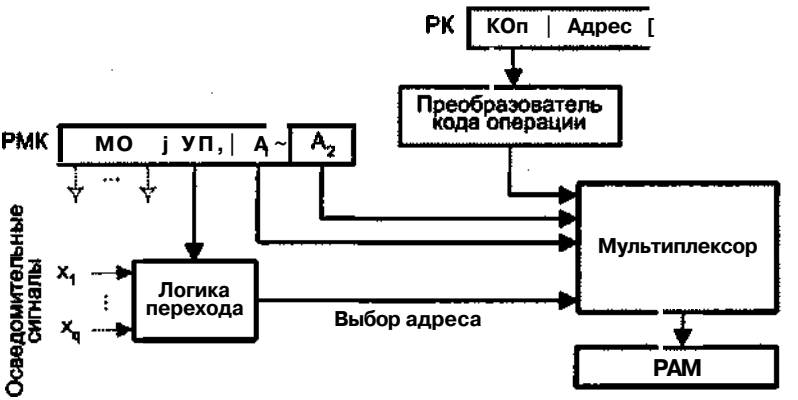


Рис. 6.13. Логика управления переходом с двумя адресными полями

С помощью мультиплексора в регистр адреса микрокоманды (РАМ) может быть загружен либо адрес, определяемый кодом операции выполняемой команды, либо содержимое одного из адресных полей микрокоманды. Выбор источника адреса осуществляется сигналом «Выбор адреса», вырабатываемым логикой перехода, на основании состояния осведомительных сигналов и поля УП микрокоманды. Если УП = 0 или УП = *i,* но $x_i = 0$, то в РАМ заносится адрес A_1 либо адрес, полученный из кода операции. В противном случае в РАМ переписывается адрес $_2$.

Более распространен вариант принудительной адресации с одним адресным полем, который уже был показан на рис. 6.12. Здесь в адресной части МКуказан адрес следующей микрокоманды, который в случае условного перехода может быть модифицирован. Возможен и иной подход, в чем-то близкий естественной адресации, когда в адресной части МК задается лишь адрес возможного перехода (рис. 6.14). При естественном следовании микрокоманд адрес очередной МК формируется путем прибавления единицы к адресу текущей микрокоманды.

Главное достоинство принудительной адресации- высокая универсальность и быстродействие. Здесь изменение участка микропрограммы не затрагивает остальных микрокоманд, а совмещение в одной МК условного перехода с формированием сигналов управления уменьшает общее время выполнения микропрограммы.

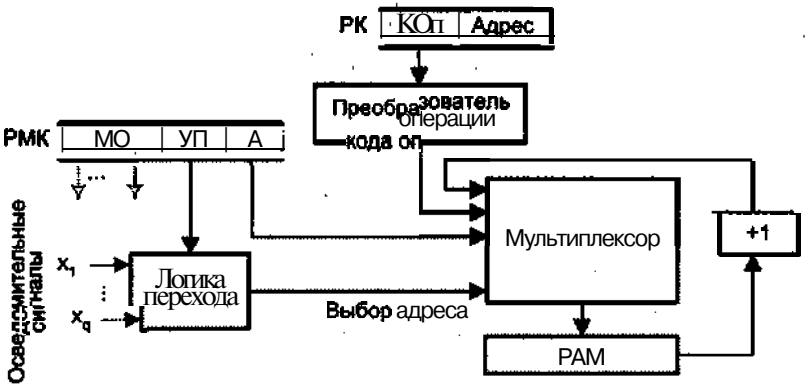


Рис. 6.14. Логика управления переходом с одним адресным полем

Основной недостаток принудительной адресации - повышенные требования к емкости памяти для хранения адресов МК:

$$E_A = N_{МК}(R_A + R_{УП}) = N_{МК}(\text{int}(\log_2 N_{МК}) \text{ int}(\log_2 L)),$$

где $R_{УП}$ — разрядность поля УП; L — общее количество осведомительных сигналов.

При естественной адресации отпадает необходимость во введении адресной части в каждую МК. Подразумевается, что микрокоманды следуют в естественном порядке и процесс адресации реализуется счетчиком адреса микрокоманды (СЧАМ). Значение СЧАМ увеличивается на единицу после чтения очередной МК.

Однако после выполнения МК с адресом A может потребоваться переход к МК с адресом B + 1. Переход может быть безусловным или зависеть от текущего значения x_i (если $x_i = 1$, то $_{МК} = A + 1$; если $x_i = 0$, то $_{МК} = B$). Для реализации условных и безусловных переходов используются специальные управляющие микрокоманды, состоящие только из двух полей: адресного поля B и поля УП, выделяющего номер условия перехода. Алгоритм выполнения управляющей МК:

если УП = 0, то СЧАМ := В;

если УП !=0, то если x_i - 1, то СЧАМ := В, иначе СЧАМ := СЧАМ + 1.

Таким образом, при естественной адресации должны применяться МК двух типов: управляющие и операционные. Операционная микрокоманда содержит только микрооперационную часть и не имеет адресной части. Тип МК задается ее первым разрядом: если МК(1) = 1, то это управляющая микрокоманда.

Структура МПА с естественной адресацией показана на рис. 6.15. Выдача сигналов управления C_1..., C_m стробируется сигналом $\overline{МК(1)}$, принимающимединичное значение при выполнении операционной МК. Дешифратор условия перехода стробируется сигналом МК(1), который равен 1 при обработке управляющей микрокоманды. Адрес следующей МК образуется на счетчике СЧАМ при выполнении микрооперации +1СЧАМ СЧАМ := СЧАМ + 1 или $П_2$СЧАМ: СЧАМ := В. Формирование адреса следующей МК описывается микропрограммой на рис. 6.16.

Достоинство естественной адресации — экономия памяти микропрограмм, а основной недостаток состоит в том, что для любого перехода требуется полный тактовый период, в то время как при принудительной адресации переход выпол-

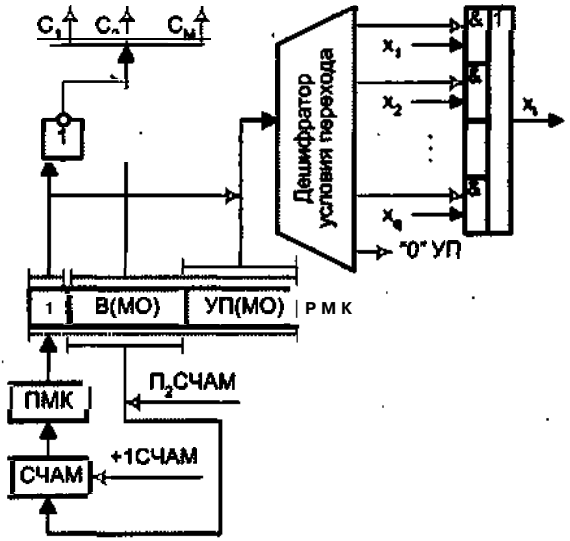


Рис. 6.15. Микропрограммный автомат с естественной адресацией

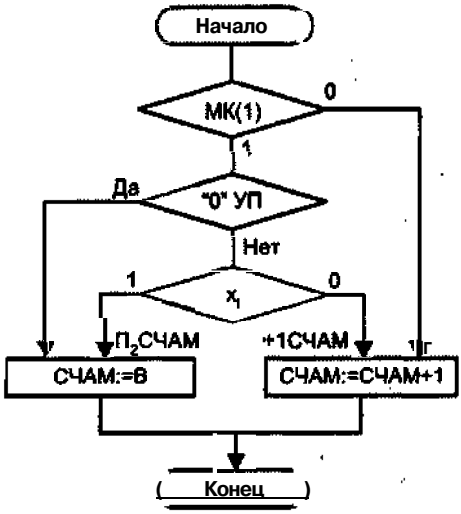


Рис. 6.16. Формирование адреса следующей микрокоманды

няется одновременно с формированием управляющих сигналов без дополнительных обращений к управляющей памяти. Кроме того, при сильно разветвленных микропрограммах требуются большие дополнительные затраты памяти:

$$\Delta E_A = (N_{УП} + N_{ВП}) \times (R_A + R_{уп}).$$

Рассмотренные способы адресации аппаратно реализует формирователь адреса микрокоманды ФАМ (см. рис. 6.8). ФАМ является механизмом управления последовательностью выполнения микрокоманд. Возрастание сложности микропрограммного обеспечения современных ВМ предопределяет необходимость расширения функциональных возможностей ФАМ.

Набор базовых функций управления, реализуемых ФАМ, включает в себя: ПРИРАЩЕНИЕ, ПЕРЕХОД ВЫЗОВ, ВОЗВРАТ, ЦИКЛ [9, 32]. Функции управления кодируются полем ФУ в составе МК (рис. 6.17) и задают алгоритмы выбора адреса очередной МК.

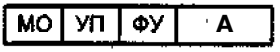


Рис. 6.17. Структура микрокоманды с выделенным полем функции управления

Обозначим: *i* - адрес МК, в которой размещена данная функция управления; АМК - адрес следующей МК; СЧЦ - счетчик количества повторений микропрограммы (циклов); x_j — значение анализируемого условия; а — адрес возврата к вызывающей микропрограмме. Охарактеризуем каждую функцию управления.

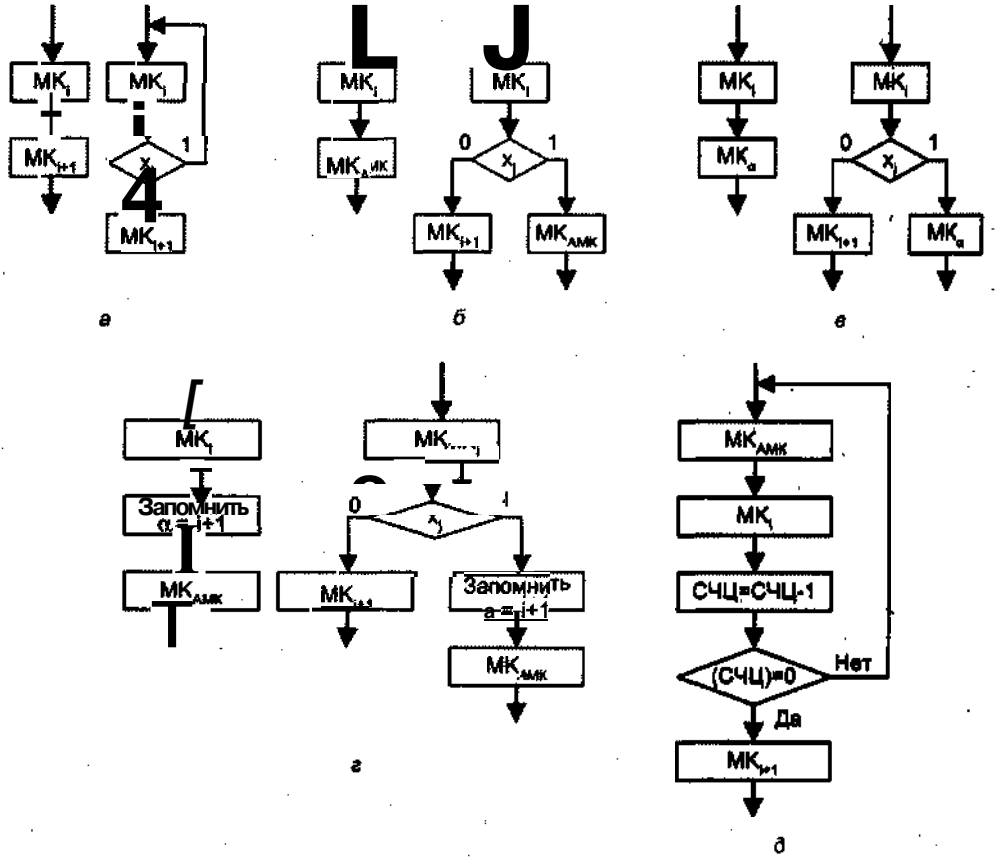


Рис. 6.18. Набор базовых функций управления микрокоманд: а - ПРИРАЩЕНИЕ; *б* - ПЕРЕХОД; в — ВОЗВРАТ; **г**— ВЫЗОВ; *д* - ЦИКЛ

ПРИРАЩЕНИЕ - обеспечивает переход к МК, записанной по адресу *i* + 1, *а условная функция* ПРИРАЩЕНИЕ — многократное повторение одной и той же МК, записанной по адресу *i* (рис. 6.18, *а).*

ПЕРЕХОД — обеспечивает переход к последовательности микрокоманд с начальным адресом АМК. В случае *условного перехода* управление передается по адресу АМК при единичном значении условия *Xj , в* противном случае выполняется МК по адресу *i* + 1 (рис. 6.18, *б).*

ВОЗВРАТ — позволяет после выполнения микропрограммы автоматически вернуться в ту точку, откуда она была вызвана (рис. 6.18, в). Использование *условной функции* ВОЗВРАТ позволяет вернуться к основной микропрограмме из различных точек внешней микропрограммы.

ВЫЗОВ — одна из основных функций, так как позволяет перейти к исполнению другой микропрограммы с начальным адресом АМК (с сохранением адреса точки перехода), как показано на рис. **6.18,** *г.*

ЦИКЛ — передает управление многократно исполняемому участку микропрограммы с начальным адресом АМК (рис. 6.18, д).

Таким образом, можно сделать вывод, что по своим возможностям базовые функции управления микрокоманд близки к командам переходов программного уровня управления.

Организация памяти микропрограмм

Функциональные возможности и структура микропрограммных автоматов в значительной степени зависят от организации памяти для хранения микропрограмм.

Основные способы организации памяти микропрограмм можно свести к следующим вариантам [3].

1. Каждое слово ПМП содержит одну микрокоманду. Это наиболее простая организация ПМП. Основной недостаток - в каждом такте работы МПА требуется обращение к памяти микропрограмм, что приводит к снижению быстродействия МПА.
2. Одно слово ПМП содержит несколько микрокоманд. В результате осуществляется одновременное считывание из ПМП нескольких **МК,** что позволяет повысить быстродействие УУ.
3. Сегментация ПМП, при которой память разделяется на сегменты, состоящие из *2^q* соседних слов, при этом адрес слова АМК разделяется на два поля: 5 и Л. Поле 5 определяет адрес сегмента, а поле А — адрес слова в сегменте. Адрес *S* устанавливается специальной микрокомандой. В последующих микрокомандах указывается только адрес слова А в сегменте. Таким образом, разрядность адресной части МК уменьшается.
4. Двухуровневая память (рис. 6.19). Первый уровень - микропамять, хранящая микрокоманды. Второй уровень — нанопамять, содержащая нанокоманды.

МК выбирается из микропамяти и служит для адресации слова нанопамяти, которое включает в себя необходимые сигналы управления. После выполнения микроопераций из микропамяти выбирается следующая МК. **Иначе** говоря, нанопамять реализует функцию шифратора управляющих сигналов (см. рис. 6.9, *в}.* Двухуровневая память рассматривается как способ для уменьшения необходимой емкости ПМП, ее использование целесообразно только при многократном повторении микрокоманд в микропрограмме.

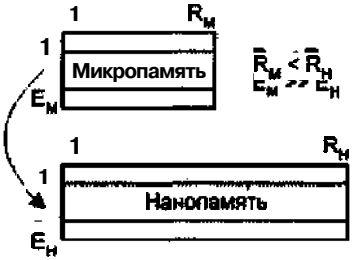


Рис. 6.19. Двухуровневая организация памяти МПА

Запоминающие устройства микропрограмм

Память микропрограмм может быть реализована запоминающими устройствами различных типов. В зависимости от типа применяемого ЗУ различают МПА со статическим и динамическим микропрограммированием [28]. В первом случае в качестве ПМП используется постоянное ЗУ (ПЗУ) или программируемая логическая матрица (ПЛМ), во втором — оперативное ЗУ.

Динамическое микропрограммирование в отличие от статического позволяет оперативно модифицировать микропрограммы УУ, меняя тем самым функциональные свойства ВМ. Основное препятствие на пути широкого.использования динамического микропрограммирования — энергозависимость и относительно невысокое быстродействие ОЗУ.

В УУ со статическим микропрограммированием более распространены ПЛМ. Программируемая логическая матрица является разновидностью ПЗУ, в котором программируются не только данные, но и адреса, благодаря чему на ПЛМ можно реализовать как память микропрограмм, так и формирователь адреса следующей микрокоманды.

ПЗУ содержит полный дешифратор с *n*-разрядным входом (адресом) и 2^n выходами. Напротив, ПЛМ имеет неполный дешифратор, количество выходов в котором меньше, чем 2^n, поэтому в ПЛМ некоторые адреса вообще не инициируют действий, тогда как другие адреса могут оказаться неразличимыми. Таким образом, возможна выборка одного слова с помощью двух или более адресов или же выборка нескольких слов с помощью одного адреса, что эквивалентно реализации функции **«ИЛИ»** от выбранных слов.

В структуре ПЛМ (рис. 6.20) выделяют три части [21]:

- буфер Б, формирующий парафазные значения $a_1, \overline{a_1}, ..., a_m, \overline{a_m}$ входных переменных $a_1, ..., a_m$;
- «И»-матрицу адресов, на выходе которой вырабатываются значения термов $Z_1, ..., Zj$
- **«ИЛИ»-матрицу** данных, на выходе которой формируются сигналы, представляющие значения выходных переменных $c_1, ..., c_n$. Матрица адресов содержит $2m$ входных цепей (горизонтальные линии) и J **m-входовых** элементов **«И»,** каждый из входов которых (вертикальные линии) может соединяться с входной цепью в точках, обозначенных крестиком. Аналогично строится матрица-данных, в которой крестиками обозначены возможные соединения входных цепей матрицы с n j-входовыми элементами «ИЛИ».

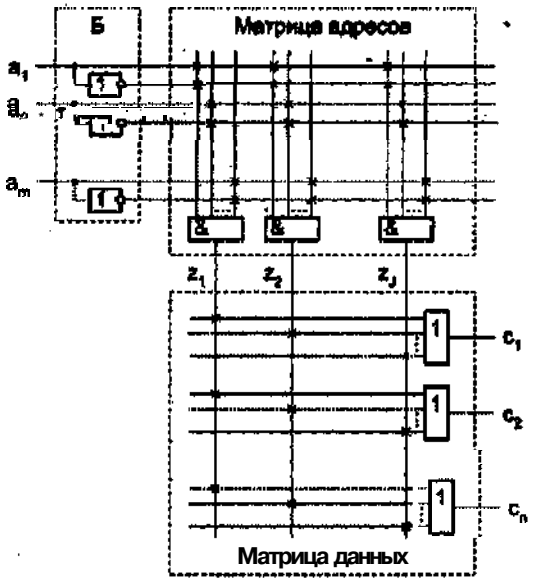


Рис. 6.20. Структура программируемой логической матрицы

Соединения в ПЛМ обеспечивают однонаправленную передачу сигналов: в матрице «И» - из любой горизонтальной цепи в вертикальную, а в матрице «ИЛИ» - из любой вертикальной цепи в горизонтальную. Такие соединения осуществляются за счет диодов или транзисторов.

Процесс установления соединений между горизонтальными и вертикальными цепями матриц называется программированием ПЛМ (программированием адресов и данных соответственно). Различают ПЛМ с масочным и электрическим программированием, а также перепрограммируемые логические матрицы. В ПЛМ первого типа информация заносится посредством подключения диодов или транзисторов. к цепям за счет металлизации соответствующих участков матриц, выполняемой через маску (шаблон). В ПЛМ с электрическим программированием либо устанавливаются нужные соединения (путем пробоя слоя диэлектрика, (р-п)-перехода), либо уничтожаются ненужные соединения (благодаря выжиганию плавких перемычек). Перепрограммируемые ПЛМ позволяют многократно переписывать информацию. При этом стирание информации производится ультрафиолетовым излучением, а запись — электрическим током.

ПЛМ более.экономны по затратам, чем ПЗУ, в трех случаях:

- при выдаче нулевого кода;
- при формировании значения функции «логическое ИЛИ» от нескольких слов;
- при записи одного и того же слова по нескольким адресам.

Чтобы иметь возможность выдать нулевой код из ПЗУ, необходимо обеспечить предварительную запись нуля в определенную ячейку. В ПЛМ для этого достаточно обращения по незапрограммированному адресу.

Во втором случае в ПЗУ опять должна задействоваться дополнительная ячейка для хранения кода, специально запрограммированного как логическое «ИЛИ»

от нескольких слов. В ПЛМ один адрес может относиться к нескольким словам, которые на выходе матрицы данных объединяются по схеме «ИЛИ».

В третьем случае в ПЗУ требуется многократная запись одного слова по всем указанным адресам. Применительно к ПЛМ это означает, что какое-то слово в матрице данных имеет адрес вида 010XX, где XX — разряды с безразличным значением (цепи соответствующих разрядов буфера не подключены к вертикальным цепям матрицы адресов).

Минимизация количества слов памяти микропрограмм

Общая задача оптимизации емкости памяти микропрограмм достаточно сложна и поэтому обычно решается в два этапа. На первом этапе минимизируется количество слов микропрограммы (количество микрокоманд), на втором этапе - разрядность микрокоманды.

Сначала рассмотрим один из возможных подходов, действующих при разбиении линейной микропрограммы на минимальное количество микрокоманд [3].

Обозначим через, - множество операндов микрооперации у; O_i - множество результатов микрооперации у.,. Сигналы управления (СУ) микроопераций у. и y_j нельзя объединить в одну микрокоманду, если имеет место один из трех случаев пересечения по данным:

$$O_i \cap I_i \neq \varnothing;$$
$$O_i \cap O_i \neq \varnothing;$$
$$I_i \cap O_i \neq \varnothing.$$

При этом говорят, что у. и у находятся в отношении w_g зависимости по данным: у. w_g y_j Графически отношение w_g отображается графом зависимости по данным (ГЗД), вершины которого соответствуют микрооперациям. Дуга (у. ,у.) в ГЗД показывает, что микрооперации у. и у находятся в отношении w_g. Пример ГЗД приведен на рис. 6.21.

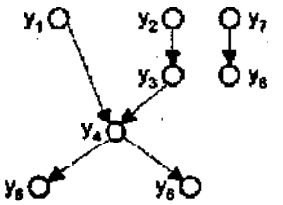


Рис. 6.21. Граф зависимости по данным

Если две микрооперации у,, y_j используют один и тот же функциональный узел ВМ, то говорят, что они находятся в отношении структурной несовместимости w_c, то есть у. w_c у Сигналы управления двух микроопераций можно объединить в одну микрокоманду, если они структурно совместимы и не находятся в отношении зависимости по данным:

$$y_i, y_j \text{ с } MK, \text{ если } (y_i \ w_c \ y_j) \& (\overline{y_i w_g} \ y_j) \neq \varnothing.$$

Пусть между **микрооперациями** $y_1, \ldots y_8$ рассматриваемого ГЗД существует следующая структурная несовместимость:

$$y_1 \; w_c \; y_7 \;,\; y_3 \; w_c y_7 \;,\; y_5 \; w_c \; y_7,$$

Минимальное количество микрокоманд в микропрограмме будет определяться длиной критического пути в графе ГЗД, то есть пути, длина которого максимальна.

На первом шаге алгоритма формируется *начальное распределение* (НР), состоящее из блоков Д .., B_M, где M- длина критического пути ГЗД. Каждый блок B_i является «заготовкой» микрокоманды Y_i , в него включаются СУ тех микроопераций, которые могут выполняться не ранее, чем по микрокоманде Y_i. Начальное распределение для нашего примера показано в табл. 6.1.

Таблица 6.1. Основные распределения микроопераций

Блок	Распределение МО			Блок	ПКР
	НР	ПР	КР		
B_1	y_1, y_2, y_7	y_2	y_2	B_1	y_2
B_2	y_3, y_8	y_1, y_3	y_3	B_2	y_3
B_3	y_4	y_4, y_7	y_4	B_3	y_4
B_4	y_5, y_6	y_5, y_6, y_8	y_5, y^*	B_4^1	y_5
				B_4^2	y_6

Так как $y_1 y_2, y_7$ независимы по данным от других микроопераций, то они могут выполняться в блоке B_1. Микрооперация y_3 зависит по данным от y_2, а y_8 — от y_7, поэтому **микрооперации** y_3 и y_8 могут выполняться не ранее, чем в блоке B_2. Основной принцип построения НР: СУ микрооперации y_n должен помещаться в блок Д, если в блоке B_{i-1} находится СУ такой микрооперации y_m, что $y_m w_g y_n$. Полученное НР (см. табл. 6.1) содержит четыре блока, что равно длине максимального пути в ГЗД (см. рис. 6.21), и определяет минимально возможное количество МК в микропрограмме.

На втором шаге алгоритма формируется *повторное распределение* (ПР), определяющее максимальный по номеру блок Д, в котором еще в состоянии выполняться микрооперация y_n. Если НР формируется прохождением графа сверху вниз, то ПР — проходом по ГЗД снизу вверх. Так, y_8 может выполняться в самом последнем блоке ПР (см. табл. 6.1), поскольку ни одна из микроопераций не зависит по данным от y_8. Длина ПР также равна длине максимального пути в ГЗД.

Исходя из НР и ПР, формируется *критическое распределение* (КР). В КР входят СУ критических микроопераций, то есть таких, которые находятся в блоках с одинаковыми номерами как в НР, так и в ПР. Так, СУ y_2 находится в блоке Д НР и в блоке B_1 ПР, следовательно, это критическая микрооперация. К критическим относятся также микрооперации y_3, y_4, y_5, y_6 (см. табл. 6.1). Кроме того, критическое распределение содержит четыре блока и является основой для формирования набора микрокоманд.

Так как КР было сформировано без учета структурной несовместимости между критическими микрооперациями, оно должно быть проверено на ее наличие.

Блок, содержащий СУ структурно несовместимых микроопераций, «расщепляется» на подблоки, внутри которых нет структурной несовместимости. Тем самым формируется *проверенное критическое распределение* (ПКР). В нашем случае существует структурная несовместимость между y_5 и y_6, поэтому блок B_4, входящий в КР, необходимо разделить на два подблока B_4^1 и B_4^2 (см. табл. 6.1).

На последнем шаге алгоритма путем размещения оставшихся СУ некритических микроопераций по блокам ПКР формируется окончательный набор микрокоманд (НМК). Каждый блок и подблок ПКР соответствует одной микрокоманде результирующего НМК. В нашем примере нужно распределить СУ микрооперации y_1, y_7, y_8. Для этого имеется следующее правило: в ПКР ищется самый первый по номеру блок, в который можно включить СУ микрооперации y_n, из последовательности блоков $B_{\text{Н}}, ..., B_{\text{К}}$, где $B_{\text{Н}}$ – блок НР и $B_{\text{К}}$ - блок ПР, содержащие СУ данной микрооперации. Если размещаемый СУ y_n не может быть включен ни в один из этих блоков из-за структурной несовместимости с уже размещенными в них СУ микроопераций, то он помещается в специально формируемый блок $B_{\text{К}}^i$ ($i = 1,2,...$). В табл. 6.2 показано применение этого правила для окончательного формирования НМК. Так, СУ микрооперации y_i должен быть включен в микрокоманду У, или в микрокоманду Y_2. СУ микрооперации y_1 включается в МК Y_1, так как он структурно совместим с СУ микрооперации y_2 принадлежит Y_1. СУ микрооперации y_7 должен быть включен в одну из микрокоманд $У_1$, К,, Y_3, однако его нельзя включить в из-за структурной несовместимости с СУ микрооперации y_2. Нельзя его поместить и в микрокоманду Y_2 вследствие структурной несовместимости с y_3. Поэтому СУ y_7 войдет в микрокоманду $У_3$.

Таблица 6.2. Формирование окончательного НМК

МК	Шаг выполнения алгоритма		
	1	Г	3
Y_1	y_1, y_2	y_1, y_2	y_1, y_2
Y_2	y_3	y_3	y_3
Y_3	y_4	y_4, y_7	y_4, y_7
Y_4	y_5	y_5	y_5, y_8
Y_5	y_6	y_6	y_6

Последней размещается СУ микрооперации y_8, и сначала может показаться, что он должен быть включен в МК Y_2, но при этом нарушается ограничение, которое накладывается зависимостью по данным: $y_7 w_g y_8$. Поэтому СУ y_8 не может войти в МК, предшествующую микрокоманде Y_3, которая содержит СУ микрооперации y_7. Для исключения подобных ошибок следует корректировать НР после каждого распределения СУ микрооперации в МК Y_i. Коррекция заключается в перемещении всех СУ микроопераций y_n, зависящих по данным от распределенного СУ микрооперации, в блок B_i НР, где номер блока равен: $i = j + 1$. Следовательно, СУ микрооперации y_8 после распределения y_7 можно включить только в МК Y_4 (см. табл. 6.2).

Минимизация разрядности микрокоманды

Пусть записанная в ПМП микропрограмма содержит m микрокоманд $Y_1, ..., Y_m$. На множестве $\{y_1,..,y_n\}$ всех микроопераций, чьи СУ входят в эти МК, задается отношение несовместимости $\overline{w_c}$, такое что:

$$y_i \overline{w_c}\ y_i \Leftrightarrow (y_i \in Y_m \rightarrow y_j \notin Y_m)(i \neq j).$$

Таким образом, несовместимыми являются микрооперации, СУ которых не встречаются вместе ни в одной микрокоманде микропрограммы.

Класс несовместимости $(KH) C_i \subseteq Y$- множество МО, все элементы которого попарно несовместимы. *Максимальный класс несовместимости* (МКН) - это такой класс, в который нельзя **бавить** ни одной микрооперации без нарушения отношения **несовместимости** w_c.

Задача минимизации разрядности микрокоманды формулируется в [3] следующим образом: найти множество классов несовместимости $C_{i_1}, ..., C_{i_r}$ такое, что

$$\bigcup_{k=1}^{H} C_{i_k} = Y; B = \sum_{k=1}^{H} \text{int}(\log_2(|C_{i_k}| + 1)) \rightarrow \min,$$

где $|C_i|$ — количество микроопераций в классе несовместимости, а выражение $\text{int}(\log_2(n+1))$ определяет минимальную разрядность поля ПМП, необходимую для кодирования n микроопераций и признака их отсутствия в конкретной МК. **параметр** $W_i = \text{int}(\log_2(|C_i| + 1))$ называют *ценой класса* C_i.

Решение ищется на множестве **МКН.** Для заданного в табл. *6.3* примера микропрограммы получается следующий набор МКН: C_1 - $\{y_1, y_7, y_{11}\}$, C_2 - $\{y_2, y_7, y_{11}\}$, $C_3 = \{y_3, y_7, y_{10}\}$, $C_4 = \{y_4, y_7, y_{10}\}$, $C_5 = \{y_4, y_9\}$, $C_6 = \{y_5, y_7, y_{10}, y_{11}\}$, $C_7 = \{y_5, y_8\}$, $C_8 = \{y_5, y_9, y_{11}\}$, $C_9 = \{y_6, y_7, y_{10}, y_{11}\}$, $C_{10} = \{y_6, y_9, y_{11}\}$.

Таблица 6.3. Набор микрокоманд

Микрокоманда	СУ микроопераций
Y_1	$y_1, y_2, y_3, y_4, y_5, y_6$
Y_2	y_3, y_7, y_8, y_9
Y_3	$y_1, y_2, y_8, y_9, y_{10}$
Y_4	y_4, y_8, y_{11}
Y_5	y_6, y_8

По набору **МКН** строится таблица покрытий, в столбце y_n которой записываются все МКН, в которые входит СУ микрооперации y_n. В табл. 6.4 имеются СУ микроопераций $y_1\ y_2, y_3\ y_8$, входящих только в один МКН. Такие микрооперации *называют различающими микрооперациями,* а соответствующие им МКН - *существенными МКН.*

Таблица 6.4. Таблица покрытий

y_1	y_2	y_3	y_4	y_5	y^*	y_m	y_6	y_9	y_{10}	y_{11}
C_1	C_2	C_3	C_4	C_6	C_9	C_1	C_7	C_5	C_3	C_1
			C_5	C_7	C_{10}	C_2		C_8	C_4	C_2
				C_8		C_4		C_{10}	C_6	C_3
						C_6			C_9	C_6
						C_9				C_8
										C_9
										C_{10}

Решением р задачи является множество МКН, включающее в себя все микро операции y_n, чьи СУ принадлежат Y. Поскольку допустимо несколько решений, то, ищется минимальное, дающее наименьшую разрядность ПМК.

Объем вычислений можно сократить путем применения эвристических правил. Так, существенные МКН (или их подклассы) C_1, C_2, C_3, C_7 (см. табл. *6.4)* должны входить в любое решение бета. Далее, если множество МКН в столбце y_i является подмножеством множества МКН в столбце y_j то столбец y_j можно удалить из таблицы покрытий, так как микрооперация y_j в любом случае покрывается меньшим числом МКН из столбца y_i (говорят, что **столбец** y_i *доминирует* над столбцом y_j). Так, в табл. 6.4 столбец y_1 доминирует над столбцами y_7 и y_{11}, столбец y_3 — над столбцом y_{10}, столбец y_8 - над столбцом y_5; следовательно, столбцы y_5, y_7, y_{10}, y_{11} можно удалить из таблицы покрытий.

Таблица 6.5. Сокращенная таблица покрытий

y_4	y_9	y_6
C_4	C_9	C_5
C_5	C_{10}	C_8
		C_{10}

После применения этих правил получается сокращенная таблица покрытий (табл. 6.5), по которой покрытия могут быть найдены, например, методом Петрика: $A' = (C_4 \vee C_5) \wedge (C_9 \vee C_{10}) \wedge (C_5 \vee C_8 \vee C_{10})$ и после приведения подобных членов и выполнения поглощений типа AB имеем: $A' = C_4C_{10} \vee C_5C_9 \vee C_5C_{10} \vee C_4C_8C_9$. Выражению A' соответствуют покрытия $\{C_4, C_{10}\}$, $\{C_4, C_8, C_9\}$, $\{C_5, C_9\}$ и $\{C_5, C_{10}\}$.

Добавив к этим покрытиям существенные МКН, получим начальное множество решений:

$\hat{P}_1 = \{C_1, C_2, C_3, C_4, C_7, C_{10}\}$;

$\hat{P}_2 = \{C_1, C_2, C_3, C_4, C_7, C_8, C_9\}$;

$\hat{P}_3 = \{C_1, C_2, C_3, C_5, C_7, C_9\}$;

$\hat{P}_4 = \{C_1, C_2, C_3, C_5, C_7, C_{10}\}$.

Решение бета j может быть избыточным, то есть некоторые микрооперации могут покрываться несколькими МКН C_i, принадлежащее бета j. Для нахождения требуемого минного решения необходимо для каждого избыточного решения бета j.

- определить множество неизбыточных решений и их цену;
- выбрать в качестве окончательного решения неизбыточное решение с минимальной ценой.

Для нахождения неизбыточных решений годится такая же процедура. Для каждого избыточного решения бета j строится таблица покрытий наподобие табл. 6.4, однако в ее столбцах записываются только МКН, соответствующее решению бета. Затем, используя описанные выше эвристические методы, получают сокращенную таблицу покрытий, из которой простым перебором или методом Петрика находятся все неизбыточные решения.

Пути повышения быстродействия автоматов микропрограммного управления

Цикл выполнения микрокоманды $T_{МК}$ можно представить в виде трех этапов:

- формирования адреса очередной микрокоманды в ФАМ ($А_i$);
- выборки по данному адресу микрокоманды из памяти микропрограмм *(В);*
- исполнения микрокоманды в операционной (ОЧ) или адресной части (АЧ) вычислительной машины *(И).*

Порядок следования этапов определяется способом соединения формирователя адреса следующей микрокоманды и памяти микропрограмм.

Структура МПА (рис. 6.22, *а)* с ФАМ и ПМП, связанными непосредственно друг с другом, не допускает совмещения этапов во времени — здесь этап $_{+1}$ начинается только после выполнения этапа $И_i$ (рис. 6.22, *б).*

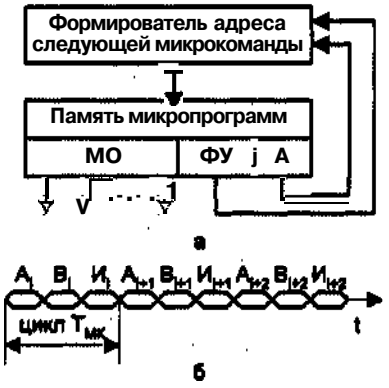


Рис. 6.22. Построение МПА **без** совмещения: а — структурная схема; *б* - временная диаграмма обработки микрокоманд

В течение цикла $T_{МК}$ все узлы заняты обработкой только одной, текущей микрокоманды с номером *i,* причем последовательный характер обработки МК не позволяет достичь высокого быстродействия:

$$T_{МК} = t_{A_i} + t_{B_i} + t_{И_i}.$$

Производительность ВМ повышается при одновременной конвейерной обработке в МПА нескольких микрокоманд, находящихся на различных этапах выпол-

нения. Для совмещения во времени этапов $A_i B_i$ и $И_i$ МПА вводятся дополнительные **запоминающие** элементы, которые хранят результаты обработки на каждом этапе.

В МПА с одним уровнем совмещения используется один запоминающий элемент - конвейерный регистр микрокоманды, подключенный к выходу памяти микропрограмм (рис. 6.23, *а*). В данной структуре реализуется одновременная обработка двух микрокоманд: в то время как исполняется находящаяся в РМК i-я микрокоманда (этап $И_i$), в ФАМ вычисляется адрес $(i + 1)$-й микрокоманды (этап $_{+1}$), и затем по этому адресу из ПМП выбирается $(i + 1)$-я микрокоманда (этап B_{i+1}). Следовательно, в МПА с одним уровнем совмещения обеспечивается параллельное выполнение этапов $_{+1}$ и B_{i+1} и этапа,__, (рис. 6.23, *б*): $T^1_{МК} = \max\{(t_A + t_B), t_И\}$.

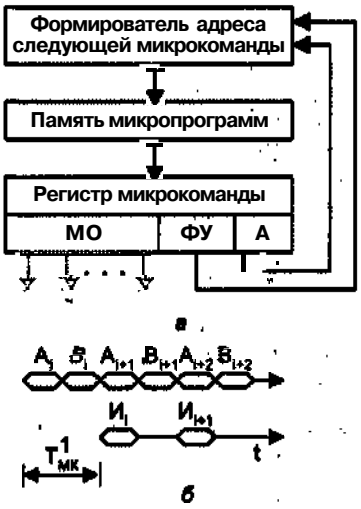


Рис. 6.23. Построение МПА с одним уровнем совмещения: а — структурная схема; *б* — временная диаграмма обработки микрокоманд

Очевидно, что при $t_И < t_A + t_B$ данной структуре наблюдается вынужденный простой операционной (или адресной) части ВМ. Этот недостаток устраняется в МПА с двумя уровнями совмещения (рис. 6.24, *а*), в котором ФАМ и ПМП связаны друг с другом двумя конвейерными регистрами - РМК и РАМ. Здесь одновременно обрабатываются три микрокоманды: в операционной или адресной части исполняется $МК_i$ (этап $И_i$) выбирается из памяти $МК_{i+1}$ (этап B_{i+1}), а в ФАМ вычисляется адрес $МК_{i+2}$ (этап A_{i+2}). Как видно из рис. 6.24, *б*, в такой структуре совмещается во времени выполнение всех трех этапов и $T^2_{МК} = \max\{t_A, t_B, t_И\}$.

Методы конвейерной обработки обеспечивают значительный выигрыш только при выполнении линейных участков микропрограммы. Если реализуемая в ФАМ функция управления является условной, а значение условия вырабатывается на текущем этапе в операционной (адресной) части, то правильный адр следующей микрокоманды может быть вычислен только по окончании этапа $И_i$, ответственного

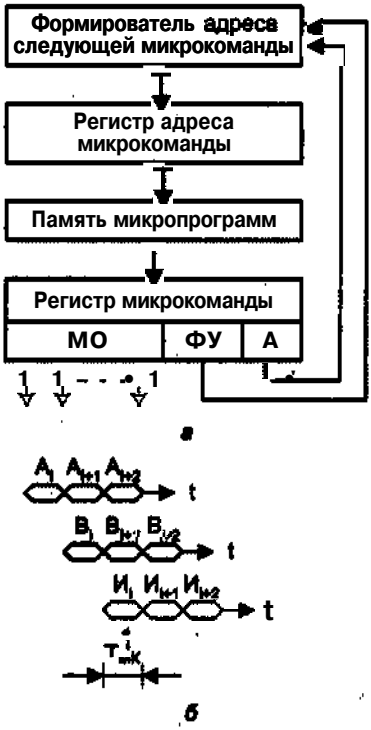


Рис. 6.24. Построение МПА с двумя уровням и совмещения: а — структурная схема; *б* - временная диаграмма обработки микрокоманд

за выработку значения условия перехода. В итоге длительность цикла для МПА с одним уровнем совмещения увеличивается до величины

$$T^1_{\text{МКУ}} = t_A + t_B + t_И,$$

а для МПА с двумя уровнями совмещения — до величины

$$T^2_{\text{МКУ}} = \max\{t_B, \ (t_И + t_A)\}.$$

а для МПА с двумя уровнями совмещения — до величины

$$T^c_{\text{МКУ}} = \max\{t_B, \ (t_И + t_A)\}.$$

Всего, в зависимости от использования регистров - регистра РМК, состоящего из адресной и микрооперационной частей, регистра РАМ, регистра состояния РГС (для хранения условия перехода), — можно сформировать 11 различных структур МПА Эти структуры, их временные диаграммы и оценки эффективности приведены в [32, с. 184-212]. Там же показано, что выбор оптимальной структуры определяется особенностями конкретной микропрограммы.

Контрольные вопросы

1. Охарактеризуйте основные функции устройства управления.
2. Какие аргументы и результат имеет функция ЦФ-ВК?
3. Какие аргументы и результат имеет функция ЦФ-ФАСК?

4. Какие аргументы и результат имеет функция ЦФ-ФИА? Сколько модифик ций она поддерживает?
5. Какие аргументы и результат имеет функция ЦФ-ВО?
6. Какие аргументы и результат имеет функция ЦФ-ИО? Сколько модификаци она поддерживает?
7. Какие этапы включаются в машинный цикл команды типа «Сложение»?
8. Какие этапы входят в машинный цикл команды типа «Запись»? Обоснуйте с сутствие одного из этапов.
9. Какие этапы входят в машинный цикл команды типа «Условный переход Обоснуйте отсутствие двух этапов.
10. Дайте характеристику входной и выходной информации модели **У** У.
11. На какие две части делится структура УУ? Что входит в состав каждой част Какое назначение имеют элементы частей УУ?
12. Обоснуйте название МПА с жесткой логикой.
13. Перечислите достоинства **и** недостатки МПА с жесткой логикой.
14. Обоснуйте название МПА с программируемой логикой. Сформулируйте до тоинства и недостатки таких МПА.
15. Дайте характеристику элементов структуры МПА с программируемой л гикой.
16. Объясните принцип управления на основе МПА с программируемой логико
17. Какие способы кодирования микрокоманд вы знаете? Перечислите их дост инства и недостатки.
18. Чем отличается принцип прямого кодирования микрокоманд от принципа ко венного кодирования?
19. **В** чем заключается суть нанопрограммирования?
20. Поясните подходы к адресации микрокоманд, охарактеризуйте их сильны и слабые стороны.
21. Какие способы организации памяти микрокоманд вы знаете, чем они обусло лены?
22. В каких случаях в МПА следует применять ПЛМ, а не ПЗУ? Ответ аргумент руйте.
23. Сформулируйте числовые параметры МПА с программируемой логикой. С чего они зависят?
24. Выберите функциональную организацию и структуру МПА с программируемо логикой. Для заданной системы команд рассчитайте числовые параметры МП
25. Выберите функциональную организацию и структуру УУ **и** МПА с програм мируемой логикой. Напишите микропрограммы реализации всех способов а ресации заданной системы команд.
26. Поясните методику минимизации количества слов памяти микрокоманд.
27. Охарактеризуйте методику минимизации разрядности микрокоманд.
28. Какие подходы к повышению быстродействия МПА с программируемой лог

Глава 7

Операционные устройства вычислительных машин

В классической фон-неймановской ВМ функция арифметической и логической обработки данных возлагается на арифметико-логическое устройство (АЛУ ч и - тывая разнообразие выполняемых операций и типов обрабатываемых данных, реально можно говорить не о едином устройстве, а о комплексе специализированных операционных устройств (ОПУ), каждое из которых реализует определенное подмножество арифметических или логических операций, предусмотренных системой команд ВМ. С этих позиций следует выделить:

- ОПУ целочисленной арифметики;
- ОПУ для реализации логических операций;
- ОПУ десятичной арифметики;
- ОПУ для чисел с плавающей запятой.

На практике две первых группы обычно объединяются в рамках одного операционного устройства. Специализированные ОПУ десятичной арифметики в современных ВМ встречаются достаточно редко, поскольку обработку чисел, представленных в двоично-десятичной форме, можно достаточно эффективно организовать на базе средств целочисленной двоичной арифметики. Таким образом, будем считать, что АЛУ образуют два вида операционных устройств: целочисленное ОПУ и ОПУ для обработки чисел в формате с плавающей запятой (ПЗ).

В минимальном варианте АЛУ должно содержать аппаратуру для реализации лишь основных логических операций, сдвигов, а также сложения и вычитания чисел в форме с фиксированной запятой (ФЗ). Опираясь на этот набор, можно программным способом обеспечить выполнение остальных арифметических и логических операций как для чисел с фиксированной запятой, так и для других форм представления информации. Следует, однако, учитывать, что подобный вариант не позволяет добиться высокой скорости вычислений, поэтому по мере расширения технологических возможностей доля аппаратных средств в составе АЛУ постоянно возрастает.

Порядок следования целевых функций полностью определяет динамику работы устройства управления и всей ВМ в целом. Этот порядок удобно задавать и отображать в виде *граф-схемы этапов* исполнения команды (ГСЭ). Как и граф-схема микропрограммы, ГСЭ содержит начальную, конечную, операторные и условные вершины. В начальной и конечной вершинах проставляется условное обозначение конкретной команды, а в условной вершине записывается логическое условие, влияющее на порядок следования этапов. В операторные вершины вписываются операторы этапов.

По форме записи оператор этапа — это оператор присваивания, в котором:

- слева от знака присваивания указывается наименование результата действий, выполненных на этапе;
- справа от знака присваивания записывается идентификатор целевой функции, определяющей текущие действия, а за ним (в скобках) приводится список исходных данных этапа.

Исходной информацией для первого этапа служит хранящийся в счетчике команд адрес A_{Ki} текущей команды K. Процесс выборки команды отображается оператором первого этапа: Ki := **ВК(A_{Ki})**.

Адрес A_{Ki} обеспечивает также второй этап, результатом которого является адрес следующей команды A_{Ki+1}, поэтому оператор второго этапа имеет вид: A_{Ki+1} := **ФАСК(A_{Ki})**.

В качестве исходных данных для третьего этапа машинного цикла выступают содержащиеся в коде текущей команды способ адресации CAi (он определяет конкретную модификацию ЦФ-ФИАО) и код адресной части A_i. Результатом становится исполнительный адрес операнда $A_{исп}$:= **ФИА(C_{Ai}, A,)**.

Полученный адрес используется на четвертом этапе для выборки операнда O, := **ВО($A_{исп}$)**.

Результат исполнения операции $РО_i$, получаемый на пятом этапе машинного цикла, зависит от кода операции i-Й команды $КОп_i$ (определяет модификацию ЦФ-ИО), кода первого операнда O_i и кода второго операнда — результата предыдущей $(i-1)$-й операции P0i-1: POi := **ИО($КОп_i$, O_i,** P0i-1).

В соответствии со структурой граф-схемы этапов все команды ВМ можно разделить на три типа:

- команды типа «Сложение» (Сл);
- команды типа «Запись» (Зп);
- команды типа «Условный переход» (УП).

Типовые граф-схемы этапов представлены на рис. 6.1.

Видно, что количество этапов в командах типа «Сл» (см. рис. 6.1, *а)* колеблется от трех (для непосредственной адресации НА) до пяти. При непосредственной адресации второй операнд записан в адресной части команды, поэтому нет необходимости в реализации устройством управления целевых функций ЦФ-ФИА, ЦФ-ВО. Количество этапов для команд типа «Зп» постоянно и равно четырем (см. рис. 6.1, *б)* — здесь отсутствует необходимость в ЦФ-ВО. Машинный цикл команд типа «УП» состоит из трех этапов (см. рис. 6.1, *в),* поскольку здесь, помимо выборки операнда, можно исключить и этап ФАСК — действия, обычно выполняемые на этом этапе, фактически реализуются на этапе ИО.

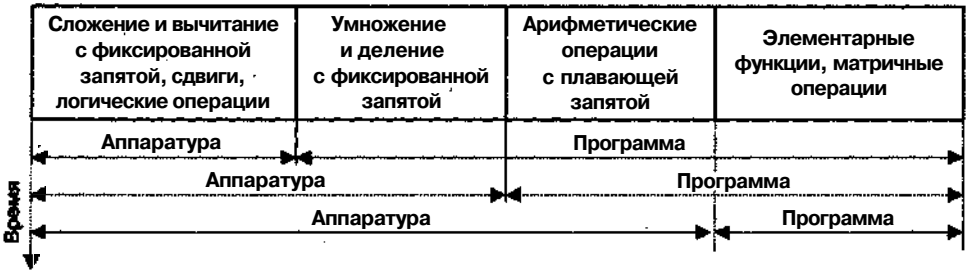


Рис. 7.1. Динамика изменения соотношения между аппаратной и программной реализациями функций АЛУ

На рис. 7.1 показана динамика изменения соотношения между аппаратной и программной реализациями функций АЛУ по мере развития элементной базы вычислительной техники. Здесь подразумевается, что по вертикальной оси откладывается календарное время.

Структуры операционных устройств

Набор элементов, на основе которых строятся структуры различных ОПУ, называется *структурным базисом*. Структурный базис ОПУ включает в себя:

- регистры, обеспечивающие кратковременное хранение слов данных;
- управляемые шины, предназначенныедля передачи слов данных;
- комбинационные схемы, реализующие вычисление функций микроопераций и логических условий по управляющим сигналам от устройства управления.

Используя методику, изложенную в [21], можно синтезировать ОПУ с так называемой канонической структурой, являющуюся основополагающей для синтеза других структур. Такая структура образуется путем замены каждого элемента реализуемой функции соответствующим элементом структурного базиса. Каноническая структура имеет максимальную производительность по сравнению с другими вариантами структур, однако по затратам оборудования является избыточной. С практических позиций больший интерес представляют два иных вида структур ОПУ: жесткая и магистральная.

Операционные устройства с жесткой структурой

В ОПУ с жесткой структурой комбинационные схемы жестко распределены между всеми регистрами. К каждому регистру относится свой набор комбинационных схем, позволяющих реализовать определенные микрооперации. Пример ОПУ с жесткой структурой, обеспечивающего выполнение операций типа «сложение», приведен на рис. 7.2.

В состав ОПУ входят три регистра со своими логическими схемами:

- регистр первого слагаемого РСл1 и схема ЛРСл1;
- регистр второго слагаемого РСл 2 и схема ЛРСл2;
- регистр суммы РСм и схема комбинационного сумматора См.

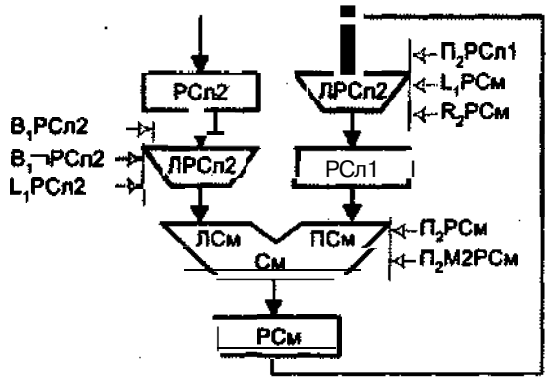


Рис. 7.2. Операционное устройство с жесткой структурой

Логическая схема ЛРСл2 реализует микрооперации передачи второго слагаемого из РСл2 на левый вход сумматора:

- прямым кодом ЛСм := РСл2 (по сигналу управления B_1РСл2);
- инверсным кодом ЛСм := ¬РСл2 (по сигналу управления B_1¬РСл2);
- со сдвигом на один разряд влево ЛСм := L1(РСл2 • 0) (по сигналу управления L1РСл2).

Логическая схема ЛРСл1 обеспечивает передачу результата из регистра РСм в регистр РСл1:

- прямым кодом РСл1 := РСм (по сигналу управления $П_2$РСл1);
- со сдвигом на один разряд влево РСл1 := L1(РСм • 0) (по сигналу управления L_1РСм);
- со сдвигом на два разряда вправо РСл1 := R2(s • s • РСм) (по сигналу управления R_1РСм).

Комбинационный сумматор См предназначен для суммирования (обычного или по модулю 2) операндов, поступивших на его левый (ЛСм) и правый (ПСм) входы. Результат суммирования заносится в регистр РСм: РСм := ЛСм + ПСм (по сигналу управления $П_2$РСм) или РСм > ЛСм ⊕ ПСм (по сигналу управления $П_2$М2РСм).

Моделью ОПУ с жесткой структурой является так называемый /-автомат, с особенностями синтеза которого можно ознакомиться в [21, 25].

Аппаратные затраты на ОПУ с жесткой структурой $C_Ж$ можно оценить по выражению

$$C_Ж = nC_T N + 3\sum_{i=1}^{N} n_i \sum_{j=1}^{K} k_{ij} + \sum_{i=1}^{N} n_i \sum_{j=1}^{K} C_j k_{ij},$$

где N— количество внутренних слов ОПУ; n_1, ..., n_N— длины слов; $n = (n1 + ... + n_N)/N$ - средняя длина слова; k_{ij} -количество микроопераций типа; $= 1,2,.... K$ (сложение, сдвиг, передача и т. п.), используемых для вычислений слов с номерами $i = 1,2, ..., N$; C_T — цена триггера; C_j — цена одноразрядной схемы для реализации микрооперации j-го типа.

В приведенном выражении первое слагаемое определяет затраты на хранение n-разрядных слов, второе — на связи регистров с комбинационными схемами,

а третье — суммарную стоимость комбинационных схем, реализующих микрооперации K типов над N словами.

Затраты времени на выполнение операций типа «сложение» в ОПУ с жесткой структурой равны

$$T_{ж} = t_{в} + t_{с} + t_{п},$$

где $t_{в}$ — длительность микрооперации выдачи операндов из регистров; $t_{с}$ — продолжительность микрооперации «сложение»; $t_{п}$ — длительность микрооперации приема результата в регистр.

Достоинством ОПУ с жесткой структурой является высокое быстродействие, недостатком — малая регулярность структуры, что затрудняет реализацию таких ОПУ в виде больших интегральных схем.

Операционные устройства с магистральной структурой

В ОПУ с магистральной структурой все внутренние регистры объединены в отдельный узел регистров общего назначения (РОН)[1], а все комбинационные схемы - в операционный блок (ОПБ), который зачастую ассоциируют с термином «арифметико-логическое устройство».

Операционный блок и узел регистров сообщаются между собой с помощью магистралей — отсюда и название «магистральное ОПУ».

Пример магистрального ОПУ представлен на рис. 7.3.

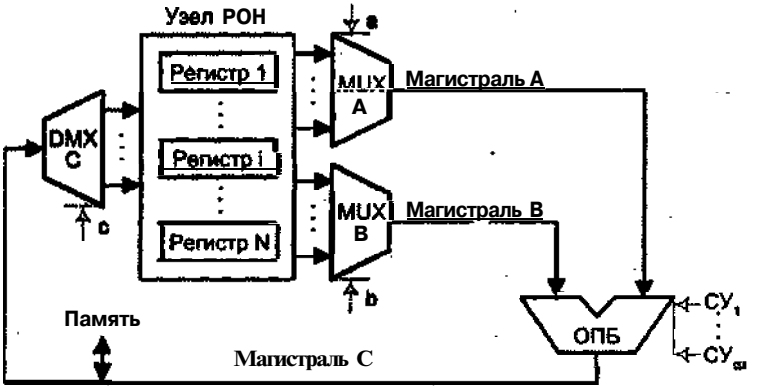


Рис. 7.3. Магистральное операционное **устройство:**

В состав узла РОН здесь входят N регистров общего назначения, подключаемых к магистралям А и В через мультиплексоры МХ А и МХ В. Каждый из мультиплексоров является управляемым коммутатором, соединяющим выход одного из РОН с соответствующей магистралью. Номер подключаемого регистра определяется адресом а или b, подаваемым на адресные входы мультиплексора из устройства управления.

[1] В операционных устройствах для обработки чисел с плавающей запятой вместо РОН часто используется отдельный узел регистров с плавающей запятой.

По магистралям A и B операнды поступают на входы операционного блока, к которым подключается комбинационная схема, реализующая требуемую микрооперацию (по сигналу управления из УУ). Таким образом, любая микрооперация ОПБ может быть выполнена над содержимым любых регистров ОПУ. Результат микрооперации по магистрали C заносится через демультиплексор DMX C в конкретный регистр узла РОН. Демультиплексор-представляет собой управляемый коммутатор, имеющий один информационный вход и N информационных выходов. Вход подключается к выходу с заданным адресом c, который поступает на адресные входы DMX C из УУ.

Моделью ОПУ с магистральной структурой является ***M*-автомат**. ***M*-автоматом** называется модель ОПУ, построенная на основе принципа объединения комбинационных схем и реализующая в каждом такте только одну микрооперацию. Синтез ***M*-автоматов** рассматривается в [22].

Используя обозначения, введенные в предыдущем разделе, выражение для оценки аппаратных затрат на магистральное ОПУ можно записать в следующем виде:

$$C_{\text{M}} = nC_{\text{T}}N + 3n(N + K) + n\sum_{j=1}^{K} C_j,$$

где первое слагаемое определяет затраты на N регистров, второе — затраты на связи узла РОН и ОПБ, а третье — суммарную цену ОПБ.

Из сопоставления выражений для затрат следует, что магистральная структура экономичнее жесткой структуры, если

$$3(N+K-M) < \sum_{i=1}^{N}\sum_{j=1}^{K} C_j K_{ij} - \sum_{j=1}^{K} C_j,$$

где $M = \sum_{i=1}^{N}\sum_{j=1}^{K} K_{ij}$ — количество микроопераций, реализуемых ОПУ с жесткой структурой.

С учетом последнего неравенства можно сформулировать следующее сильное условие экономичности магистральных структур:

$$M > N + K.$$

Затраты времени на сложение в магистральных ОПУ больше, чем в ОПУ с жесткой структурой:

$$T_{\text{M}} = t_{\text{B}} + t_{\text{C}} + t_{\text{П}} + t_{\text{MUX}} + t_{\text{DMX}} = t_{\text{Ж}} + t_{\text{MUX}} + t_{\text{DMX}},$$

где t_{MUX} — задержка на подключение операндов из РОН к ОПБ; t_{DMX} — задержка на подключение результата к РОН.

Основным достоинством магистральных ОПУ является высокая универсальность и регулярность структуры, что облегчает их реализацию на кристаллах ИС. Вообще говоря, магистральная структура ОПУ в современных ВМ является превалирующей.

Классификация операционных устройств с магистральной структурой

Магистральные ОПУ классифицируют по виду и количеству магистралей, организации узла РОН, типу ОПБ.

Магистрали ОПУ могут быть однонаправленными и двунаправленными, соответственно обеспечивающими передачу данных в одном или двух различных направлениях. Типичным режимом работы магистрали является разделение времени, при котором магистраль используется для передачи функционально разнотипных данных в различные моменты времени.

По функциональному назначению выделяют:

- *магистрали внешних связей,* соединяющих ОПУ с памятью и каналами ввода/вывода ВМ;
- *внутренние магистрали ОПУ,* отвечающие за связь между узлом РОН и операционным блоком.

Количество магистралей внешних связей зависит от архитектуры конкретной ВМ и обычно не превышает двух для внешних связей и трех — для внутренних.

Структуратрехмагистрального ОПУ представлена на рис. 7.4, *а,* а соответствующая ему микропрограмма выполнения операции типа «сложение» — на рис. 7.4, *б.*

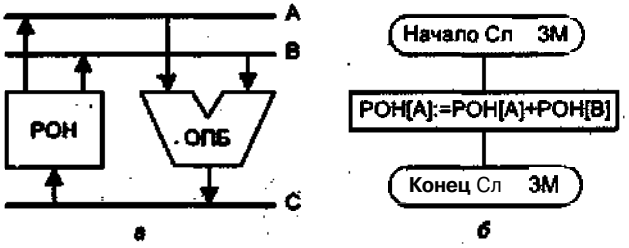


Рис. 7.4. Трехмагистральное ОПУ: а — структура; *б* — микропрограмма сложения

Данный вариант характеризуется наибольшим быстродействием: выборка операндов из РОН, выполнение микрооперации суммирования и запись результата в РОН — все эти действия производятся за один такт. Основной недостаток трехмагистральной организации - большая площадь, занимаемая магистралями на кристалле БИС (от 0,16 до 0,22 от площади кристалла).

Двухмагистральная организация при меньшей площади, покрываемой магистралями (от 0,06 до 0,19 от площади кристалла), требует введения как минимум одного буферного регистра (БР), предназначенного для временного хранения одного из операндов (рис. 7.5, *а),* при этом операция сложения будет выполняться уже за два такта (рис. 7.5, б):

- Такт 1: загрузка БР одним из операндов.
- Такт 2: выполнение микрооперации в ОПБ над содержимым БР и одного из РОН; запись результата в РОН.

Наконец, организация ОПУ на основе только одной магистрали (рис. 7.6, *а)* минимизирует расходы площади (от 0,03 до 0,09 от площади кристалла).

В одномагистральном ОПУ, вместе с тем, возникает необходимость введения не менее двух буферных регистров БР1, БР2, и длительность операции возрастает до трех Тактов (рис. 7.6, б):

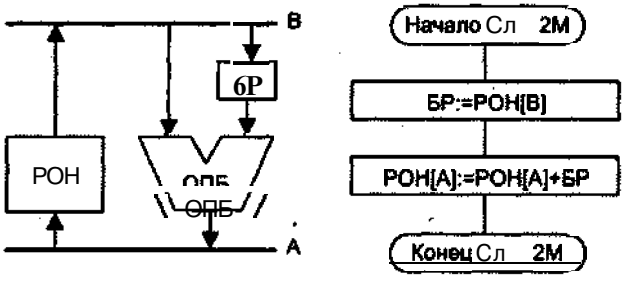


Рис. 7.5. **Двухмагистральное** ОПУ: *а* — структура; *б* — микропрограмма сложения

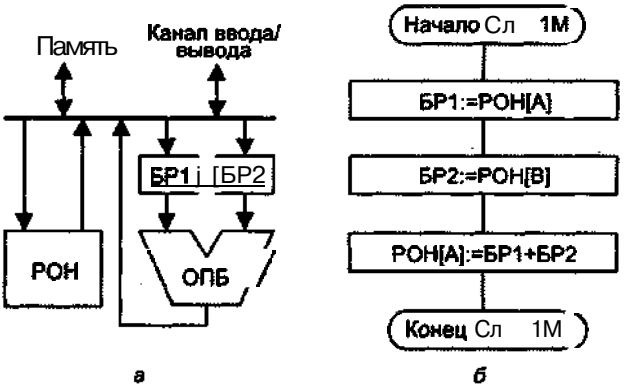


Рис. 7.6. **Одномагистральное** ОПУ: а — структура; ***б*** **—**микропрограмма сложения

- Такт 1: загрузкаБР1 одним из операндов.
- Такт 2: загрузка БР2 вторым операндом.
- Такт 3: выполнение микрооперации в ОПБ над содержимым БР1 и БР2; запись результата в один из РОН.

Организация узла РОН магистрального операционного устройства

Количество регистров в узле РОН магистрального ОПУ обычно превышает тот минимум, который необходим для реализации универсальной системы операций. Избыток регистров используется:

- *для* хранения составных частей адреса (индекса, базы);
- в качестве буферной, сверхоперативной памяти для повышения производительности ВМ за счет уменьшения требуемых пересылок между основной памятью и ОПУ.

Количество регистров колеблется в среднем от 8 до 16, иногда может достигать 32-64. В процессорах с сокращенным набором команд количество РОН доходит до нескольких сотен.

Организация узла РОН может обеспечивать одноканальный или двухканальный доступ как по входу (записи), так и по выходу (считыванию). В первом случае к входу узла подключается один демультиплексор, а к выходу — один мультиплексор. Во втором случае доступ осуществляется с помощью двух демультиплексоров и (или) двух мультиплексоров. Двухканальный доступ повышает быстродействие ОПУ, так как позволяет обратиться параллельно к двум регистрам.

Организация операционного блока магистрального операционного устройства

Тип операционного блока (ОПБ) определяется способом обработки данных. Различают ОПБ последовательного и параллельного типа.

В *последовательном операционном блоке* (рис. 7.7) операции выполняются побитово, разряд за разрядом.

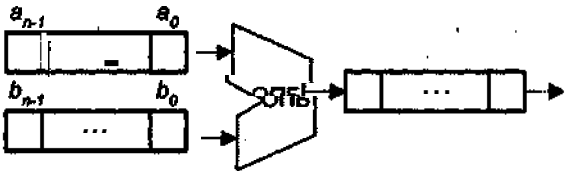


Рис. 7.7. Последовательный операционный блок

Бит переноса, возникающий при обработке *i*-го разряда операндов, подается на вход ОПБ и учитывается при обработке (*i*+ 1)-го разряда операндов. Результат побитово заносится в выходной регистр, предыдущее содержимое которого перед этим сдвигается на один разряд. Таким образом, после *n* циклов в выходном регистре формируется слово результата, где каждый разряд занимает предназначенную для него позицию.

При *параллельной организации операционного блока* (рис 7.8) все разряды операндов обрабатываются одновременно. Внутренние переносы обеспечиваются схемой ОПБ. Более подробно возможности организации переносов рассматриваются позже.

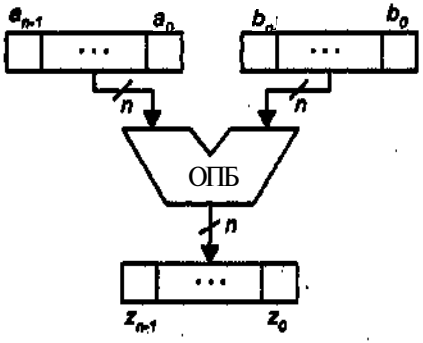


Рис. 7.8. Параллельный операционный блок

Реализация эффективной системы переносов в рамках «длинного» слова сопряжена с определенными аппаратурными издержками, поэтому на практике часто используют параллельно-последовательную схему ОПБ. В ней слово разбивается на группы по 2, 4 или 8 разрядов, обработка всех разрядов внутри группы осуществляется параллельно, а сами группы обрабатываются последовательно.

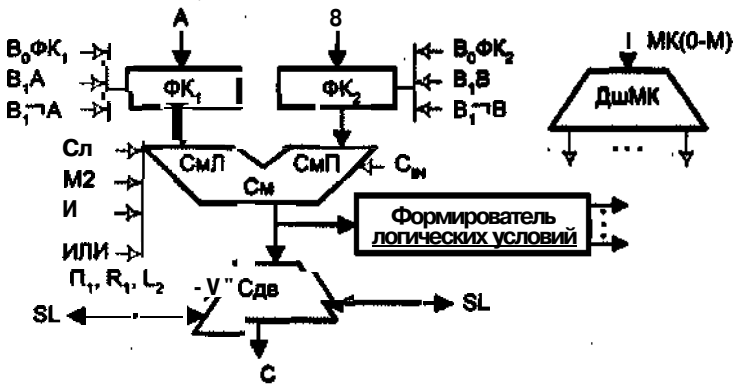


Рис. 7.0. Обобщенная схема операционного блока

Обобщенная схема ОПБ приведена на рис. 7.9. В нее входят: дешифратор микрокоманды ДшМК, формирователи кодов ФК1 и , многофункциональный сумматор См, сдвигатель Сдв и формирователь логических условий (ЛУ).

Дешифратор микрокоманды вырабатывает внутренние сигналы управления для элементов ОПБ. Он введен в схему с целью минимизации количества связей, требуемых для передачи сигналов управления из УУ.

Формирователи кодов ФК1 и $ФК_2$ служат для формирования прямых и инверсных кодов операндов, поступающих по магистралям А и В. Они реализуют следующий набор микроопераций:

$B_0ФК_1$: СмЛ := 0; $B_0ФК_2$: СмП := 0;
B_1A: СмЛ > А; B_1B: СмП := В;
$B_1\neg A$: СмЛ := -А; $B_1\neg B$: СмП := $\neg$В.

Многофункциональный сумматор выполняет микрооперации арифметического сложения (с учетом переноса C_{IN}), сложения по модулю два, логического сложения и логического умножения кодов на левом и правом входах:

Сл: См := СмЛ + СмП + C_{IN};
М2: См := СмЛ $\oplus$ СмП;
И: См := СмЛ л СмП;
ИЛИ: См := СмЛ v СмП.

Формирователь логических условий на основе анализа кода на выходе См вырабатывает значения ознакомительных сигналов, передаваемых в УУ машины. Осведомительными сигналами могут быть: признак знака S, признак переполнения V, признак нулевого значения результата Z и т. п.

Сдвигатель служит для выполнения микроопераций сдвига кода на выходе См:

$П_1$: C := См;
R_1: C := R1(SL • См), SR := См(n);
L_1: C := L1(См • SR), SL := См(0).

Микрооперация П1 обеспечивает передачу результата на магистраль С без сдвига. По ходу микрооперации R результат сдвигается на один разряд вправо, при этом в освобождающийся старший разряд заносится значение с внешнего контакта **SL**, а выдвигаемый (младший) разряд сумматора посылается на внешний контакт SR

В микрооперации L1 результат сдвигается на один разряд влево. Здесь в освобождающийся младший разряд заносится значение с внешнего контакта SR, а выдвигаемый (старший) разряд См передается на внешний контакт SL

Базис целочисленных операционных устройств

Для большинства современных ВМ общепринятым является такой формат с фиксированной запятой (ФЗ), когда запятая фиксируется справа от младшего разряда кода числа. По этой причине соответствующие операционные устройства называют целочисленными ОПУ. В форме с ФЗ могут быть представлены как числа без **знака**, когда все *n* позиций числа отводятся под значащие цифры, так и со знаком. В последнем случае старший *(n* - 1)-й разряд числа занимает знак числа (0 — плюс, 1 - минус), а под значащие цифры отведены разряды с (*n* - 2)-го по 0-й. При записи отрицательныхчисел используется дополнительный код, который для **числа** *N* получается по следующей формуле:

$$N_Д = 2^n - N = (2^n - 1) - N + 1 = \overline{N} + 1.$$

Если исключить логические операции, которые рассматриваются отдельно, целочисленное ОПУ должно обеспечивать выполнение следующих арифметических операций над числами без знака и со знаком:

- **сложение/вычитание;**
- **умножение;**
- **деление.**

Сложение и вычитание

На рис. 7.10 приводятся примеры сложения целых чисел, представленных в дополнительном коде (напомним, что при сложении в *дополнительном коде* знаковый разряд участвует в операции наравне с цифровыми).

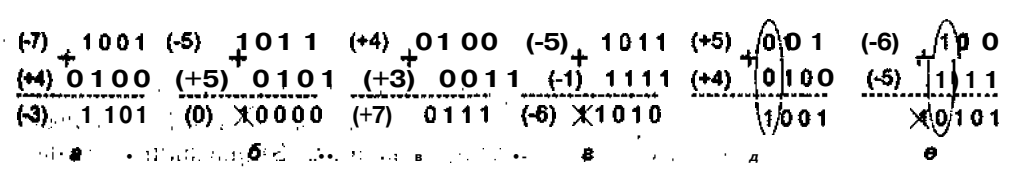


Рис. 7.10. Примеры выполнения операции сложения в дополнительном коде:
***а, б, в, г* — сложение без возникновения переполнения;**
***д, в* — сложение с переполнением**

При сложении n-разрядных двоичных чисел (бит знака и *n* - 1 значащих цифр) возможен результат, содержащий *n* значащих цифр. Эта ситуация известна как *переполнение.* «Лишний» бит занимает позицию знака, что приводит к некорректности результата. Естественно, что ОПУ должно обнаруживать факт переполнения и сигнализировать о нем. Для этого используется следующее правило: *если суммируются два числа и они оба положительные или оба отрицательные, переполнение имеет место тогда а только тогда, когда знак результата противополо жен знаку слагаемых.* Рисунки 7.10, *д* и 7.10, *е* показывают примеры переполнения. Обратим внимание, что переполнение не всегда сопровождается переносом из знакового разряда.

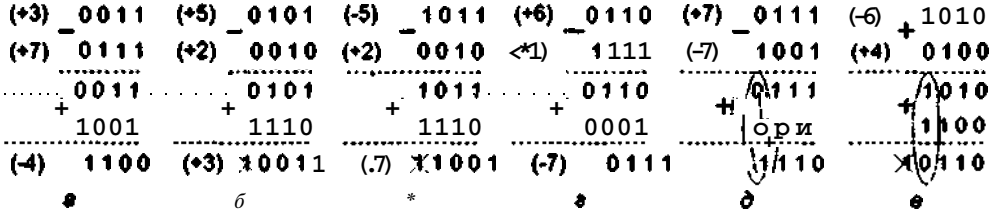


Рис. 7.11. Примеры выполнения операции вычитания в дополнительном коде: а, *б, в, г* - вычитание без возникновения переполнения; л, в — вычитание с переполнением

Вычитание выполняется в соответствии с правилом: *для вычитания одного числа (вычитаемого) из другого (уменьшаемого) необходимо взять дополнение вычитаемого и прибавить его к уменьшаемому. Под дополнением здесь понимается вы*читаемое с противоположным знаком, представленное в дополнительном коде. Вычитание иллюстрируется примерами (рис. 7.11). Два последних примера (см. рис. 7.11, *д* и 7.1 *\е)* демонстрируют ранее рассмотренное правило обнаружения переполнения.

Чтобы упростить обнаружение ситуации переполнения, часто применяется так называемый *модифицированный дополнительный код,* когда для хранения знака отводятся два разряда, причем оба участвуют в арифметической операции наравне с цифровыми разрядами. В нормальной ситуации оба знаковых разрядах содержат одинаковые значения. Различие в содержимом знаковых разрядов служит признаком возникшего переполнения (рис. 7.12).

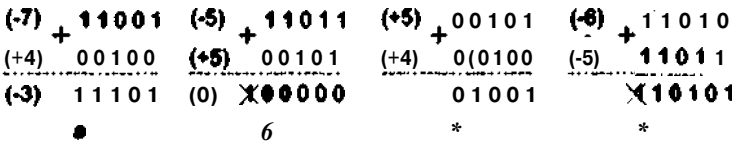


Рис. 7.12. Примеры выполнения операции сложения в модифицированном дополнительном коде: *в, б* — переполнения нет; в, г — возникло переполнение

На рис. 7.13 показана возможная структура операционного блока для сложения и вычитания чисел со знаком в формате с фиксированной запятой. Центральным звеном устройства является n-разрядный двоичный сумматор. Операнд *А* поступает на вход сумматора без изменений. Операнд *В* предварительно пропускается

через схемы сложения по модулю 2, поэтому вид кода *B,* поступающего на другой вход сумматора, зависит от выполняемой операции. Если задана операция сложения (управляющий код 0), то результат на выходе ОПБ определяется выражением $S.=A + B$. При операции вычитания (управляющий код 1) на вход сумматора подаются инверсные значения всех разрядов *B,* и, кроме того, на вход переноса в младший разряд сумматора C_{IN} поступает 1. В итоге на выходе ОПБ будет $S = A + B + 1;$ что соответствует прибавлению к *A* числа *B c* противоположным знаком, то есть вычитанию.

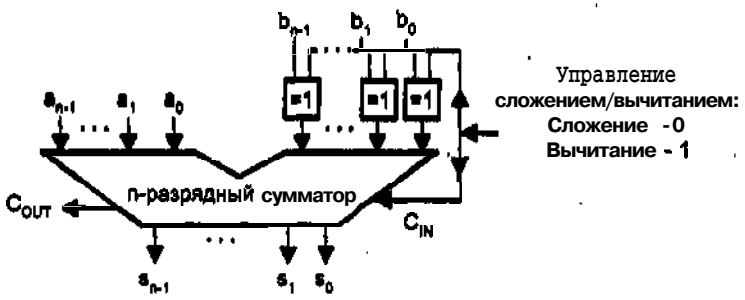


Рис. 7.13. Структура операционного блока для сложения и вычитания

На рис. 7.13 не показана схема формирования признака переполнения *V,* который согласно описанным ранее правилам определяется логическим выражением

$$V = \overline{a_{n-1}}\ л\ \overline{b_{n-1}}\ л\ s_{n-1} \vee a_{n-1}\ л\ b_{n-1}\ л\ \overline{s_{n-1}}\ .$$

Целочисленное умножение

По сравнению со сложением и вычитанием, умножение — более сложная операция, как при программном, так и при аппаратном воплощении. В ВМ применяются различные алгоритмы реализации операции умножения и, соответственно, несколько схем построения операционных блоков, обеспечивающих выполнение операции умножения.

Традиционная схема умножения похожа на известную из школьного курса процедуру записи «в столбик». Вычисление произведения $P\ (p_{2n-1}p_{2n-2} \ldots p_1p_0)$ *двух* n-разрядных двоичных чисел без знака $A\ (a_{n-1}a_{n-2} \ldots a_1\ a_0)$ и $B\ (b_{n-1}b_{n-2} \ldots b_1b_0)$ сводится к формированию частичных произведений (ЧП) W_i по одному на каждую цифру множителя, с последующим суммированием полученных ЧП. Перед суммированием каждое частичное произведение должно быть сдвинуто на один разряд относительно предыдущего согласно весу цифры множителя, которой это ЧП соответствует. Поскольку операндами являются двоичные числа, вычисление ЧП упрощается - если цифра множителя b_i равна 0, то Wi тоже равно 0, а при b_i - 1 частичное произведение равно множимому *(Wi-A).* Перемножение двух n-разрядных двоичных чисел *P - A x B* приводит к получению результата, содержащего 2n битов. Таким образом, алгоритм умножения предполагает последовательное выполнение двух операций - сложения и сдвига (рис. 7.14). Суммирование ЧП обычно производится не на завершающем этапе, а по мере их получения. Это по-

зволяет избежать необходимости хранения всех ЧП, то есть сокращает аппаратурные издержки. Согласно данной схеме, устройство умножения предполагает наличие регистров множимого, множителя и суммы частичных произведений, а также сумматора ЧП и, возможно, схем сдвига, если операция сдвига не реализована иным способом, например за счет «косой» передачи данных между узлами умножителя.

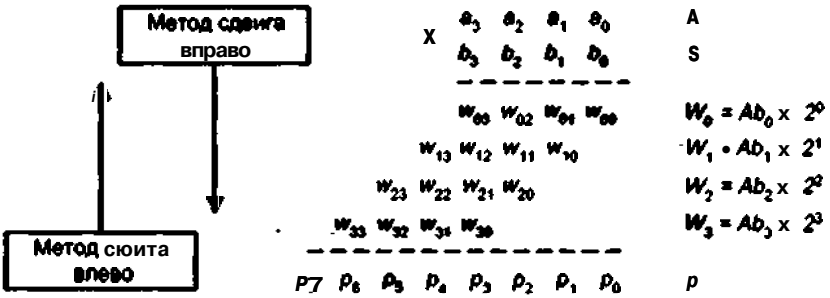


Рис. 7.14. Общая схема умножения со сдвигом суммы частичных произведений влево или вправо

В зависимости от способа получения суммы частичных произведений (СЧП) возможны четыре варианта реализации «традиционной» схемы умножения [10]:

1. Умножение, начиная с младших разрядов множителя, со сдвигом суммы частичных произведений вправо и при неподвижном множимом.
2. Умножение, начиная со старших разрядов множителя, при сдвиге суммы частичных произведений влево и неподвижном множимом.
3. Умножение, начиная с младших разрядов множителя, при сдвиге множимого влево и неподвижной сумме частичных произведений.
4. Умножение, начиная со старших разрядов множителя, со сдвигом множимого вправо и при неподвижной сумме частичных произведений.

Варианты со сдвигом множимого на практике не используются, поскольку для их реализации регистр множимого, регистр СЧП и сумматор должны иметь разрядность 2n, поэтому остановимся на вариантах 1 и 2. Первый из них назовем *алгоритмом сдвига вправо, а* второй *— алгоритмом сдвига влево*.

Умножение чисел без знака

Общую процедуру традиционного умножения сначала рассмотрим применительно к числам без знака, то есть таким числам, в которых все *n* разрядов представляют значащие цифры.

Алгоритм сдвига вправо

Алгоритм сводится к следующим шагам:

1. Исходное значение суммы частичных произведений принимается равным нулю.
2. Анализируется очередная цифра множителя (анализ начинается с младшей цифры). Если она равна единице, то к СЧП прибавляется множимое, в противном случае (цифра равна нулю) прибавление не производится.

3. Выполняется сдвиг суммы частичных произведений вправо на один разряд,
4. Пункты 2 и 3 последовательно повторяются для всех цифровых разрядов множителя.

Процедура умножения иллюстрируется примером вычисления произведения 10x11 (рис. 7.15)

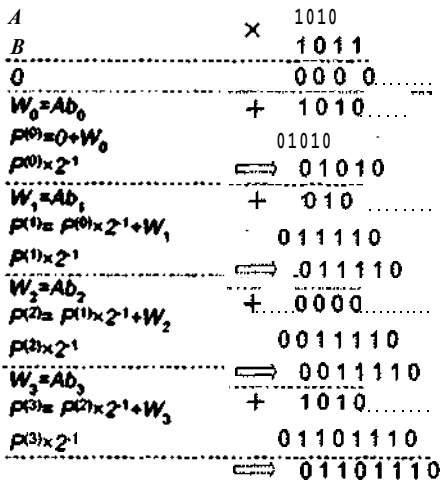


Рис. 7.15. Пример умножения со сдвигом суммы частичных произведений вправо

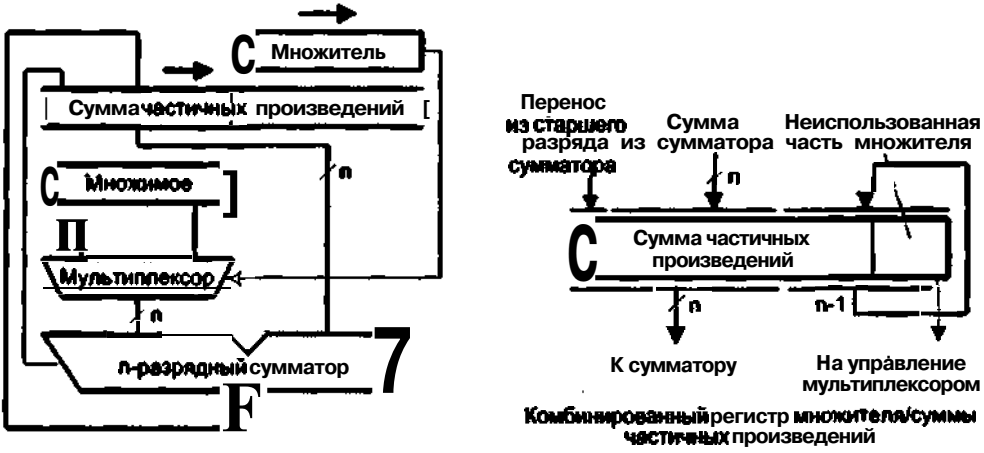


Рис. 7.16. Схема устройства умножения по алгоритму правого сдвига

Первоначально множимое и множитель заносятся в n-разрядные регистры множимого (РМт) и множителя (РМт) соответственно, а все разряды 2n-разрядного регистра суммы частичных произведений (РЧП) устанавливаются в 0. Умножение происходит за n шагов. На каждом шаге, в зависимости от состояния младшего разряда регистра множителя, управляющего мультиплексором, на один из входов

«-разрядного сумматора подается либо множимое, либо 0. На второй вход поступает содержимое *n* старших разрядов РЧП. Новое частичное произведение из сумматора пересылается в старшие разряды РЧП. Далее содержимое РЧП сдвигается на один разряд вправо, причем в освободившийся старший разряд регистра заносится значение переноса из старшего разряда сумматора. Поскольку мультиплексор управляется младшим разрядом РМт, то содержимое этого регистра также сдвигается на один разряд вправо. Описанная последовательность повторяется *n* раз. Более экономичным в плане аппаратуры является иное решение, когда вместо двух регистров - n-разрядного РМт и 2м-разрядного РЧП - используется один комбинированный 2л-разрядный регистр (показан на рис 7.16 справа). Множитель первоначально заносится в младшие *n* разрядов этого регистра, а старшие разряды обнуляются. По мере сдвигов вправо младшие, уже проанализированные разряды множителя выталкиваются из регистра, освобождая место для очередной цифры СЧП. Обычно такой регистр строится из двух n-разрядных регистров, объединенных цепями сдвига. Дополнительно отметим, что если очередная цифра множителя равна 1, то для вычисления суммы ЧП требуются операции сложения и сдвига, а при нулевой цифре множителя в принципе можно обойтись без сложения, ограничившись только сдвигом. Это, естественно, требует некоторого видоизменения схемы.

Алгоритм сдвига влево

Процедура традиционного умножения со сдвигом влево включает в себя следующие шаги:

1. Исходное значение суммы частичных произведений принимается равным нулю.
2. Анализируется очередная цифра множителя (анализ начинается со старшей цифры). Если она равна единице, то к СЧП прибавляется множимое, в противном случае (цифра равна нулю) прибавление не производится.
3. Выполняется сдвиг суммы частичных произведений влево на один разряд.
4. Пункты 2 и 3 последовательно повторяются для всех цифровых разрядов множителя.

На рис. 7.17 приведен пример умножения со сдвигом влево(10х 11).

Описанная процедура может быть реализована с помощью схемы, показанной на рис. 7.18.

К преимуществу алгоритма сдвига влево следует отнести то, что он позволяет совмещать во времени операции сложения и сдвига. Однако, по сравнению с алгоритмом сдвига вправо, он имеет и ряд недостатков. В первую очередь, СЧП и множитель не могут совместно использовать один и тот же регистр. Для реализации алгоритма требуется 2х-разрядный сумматор. Кроме того, схема со сдвигом влево неудобна при выполнении умножения над числами с разными знаками.

Тем не менее окончательный выбор между алгоритмами сдвига вправо или влево не однозначен. Так, если в результате умножения требуется результат только одинарной длины, то вариант со сдвигом влево может оказаться в аппаратурном плане выгоднее, поскольку не принуждает вводить дополнительные цепи сдвига. Конечный выбор определяется соотношением затрат оборудования на реализацию цепей сдвига и дополнительных разрядов сумматора.

A	×	1010
B		1011
0		00 00
0×2^1	⟸	00000
$W_0 = Ab_2$	+	1010
$P^{(0)} = 0 + W_0$		01010
$P^{(0)} \times r^{-1}$	⟸	01010o
$W_1 = Ab_2$	+	0000
$P^{(1)} = P^{(0)} \times 2^1 + W_1$		010100
$P^{(1)} \times 2^1$	⟸	0101000
$W_2 = Ab_1$	+	1010
$P^{(2)} = P^{(1)} \times 2^1 + W_2$		0110010
$P^{(2)} \times 2^1$	⟸	01100100
$W_3 = Ab_0$	+	1010
$P = P^{(3)} = P^{(2)} \times 2^1 + W_3$		01011110

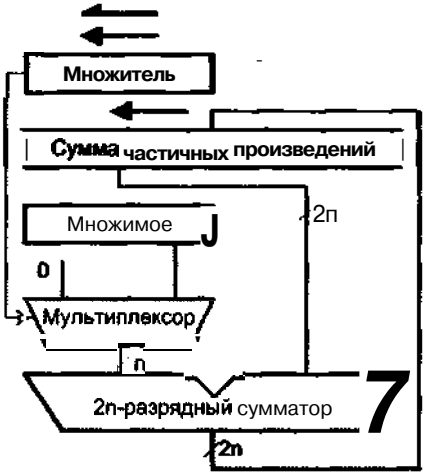


Рис. 7.17. При *дений* влево

Рис. 7.18. Схема устройства умножения по алгоритму левого сдвига

У множение чисел со знаком

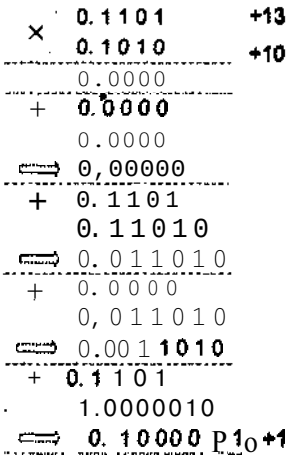


Рис. 7.19. Умножение чисел при положительных сомножителях

тельного числа в дополнительном коде сводится к инвертированию всех цифровых разрядов числа, представленного в прямом коде, и прибавлению единицы к младшему разряду получившегося после инвертирования обратного кода По этой причине более предпочтительны варианты, не требующие преобразования сомножителей и обеспечивающие вычисления непосредственно в дополнительном коде.

Задержимся на особенностях операции умножения при различных сочетаниях знаков сомножителей. Первая из них проявляется при выполнении операции арифметического сдвига вправо для суммы частичных произведений — освободившиеся при сдвиге цифровые позиции должны заполняться не нулем, а значением знакового разряда сдвигаемого числа. Здесь, однако, следует учитывать, что это правило заполнения освободившихся цифровых разрядов начинает действовать лишь с момента, когда среди анализируемых разрядов множителя появляется первая единица.

Множимое произвольного знака, множитель положительный

Пример для положительных сомножителей (Л >= О, B >= 0) уже был рассмотрен. Б случае отрицательного множимого процедура умножения протекает аналогично , с учетом сделанного замечания об арифметическом сдвиге СЧП, что подтверждает пример, приведенный на рис. 7.20.

Поскольку результат умножения отрицательный, он получается в дополнительном коде.

Множимое произвольного знака, множитель отрицательный

Так как множитель отрицателен, он записывается в дополнительном коде: $[B]_{\text{д}}$ $2^{n}-|B|$, и в цифровых разрядах кода будет представлено число $2^{n1-}-\backslash B\backslash$. При типовом умножении (как в случае $B > 0$) получим $P = A$ $(2^{n-1} - |B|) - = -\backslash B\backslash * A + A * 2^{n-1}$. Псевдопроизведение P больше истинного произведения P на

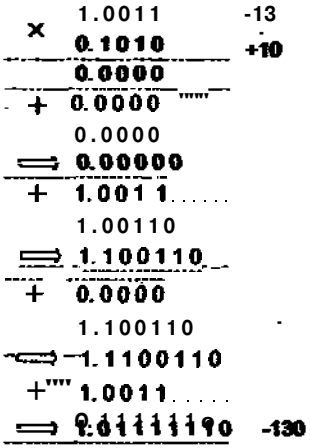


Рис. 7.20, Умножение чисел при отрицательном множимом и положительном множителе

величину $A \times 2^{n}1$, что и необходимо учитывать при формировании окончательного результата. Для этого перед последним сдвигом из полученного псевдопроизведения необходимо вычесть избыточный член. На рис. 7.21 и 7.22 приведены примеры умножения положительного и отрицательного множимого на отрицательный множитель, в которых видна упомянутая коррекция результата

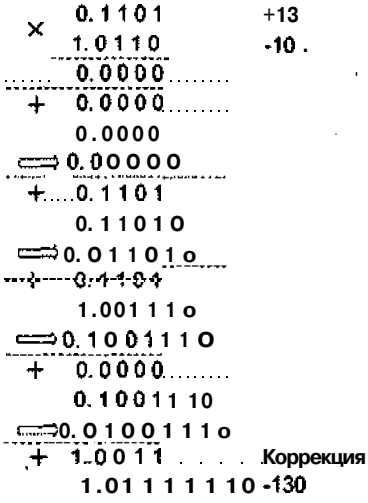


Рис. 7.21. Умножение чисел при положительном множимом и отрицательном множителе

Рассмотренные процедуры умножения чисел со знаком в принципе могут быть реализованы с помощью ранее рассмотренного устройства (см. рис. 7,16). На практике для перемножения чисел со знаком применяют иные алгоритмы. Наиболее распространенным из них является алгоритм Бута, имеющий дополнительное преимущество, — он ускоряет процесс умножения по сравнению с рассмотренным ранее. Этот алгоритм будет рассмотрен ниже.

```
      1.00 11          -13
  ×   1.0110           -10
      -------
      0.0000
     0.0000
    0.0000
   0.00000
  + 1.0011
    -------
    1.001 10
   1.1001 10
  + 1.0011
    -------
    0.110010
   1.0110010
  + 0.0000
    -------
    1.0110010
   1.1011 0010   Коррекция
  + 0.1101       +130
    -------
    0.1000001o
```

Рис. 7.22. Умножение чисел при отрицательном множимом и отрицательном множителе

Умножение целых чисел и правильных дробей

Рассмотренные алгоритмы относились к представлению чисел с фиксированной запятой, то есть, как это принято в большинстве ВМ, к целым числам. При перемножении чисел со знаком необходимо принимать во внимание, что произведение двух n-разрядных чисел со знаком (знак *n-1* значащий разряд) может иметь *2(n*-1) значащих разрядов и для его хранения обычно используют регистр двойной дли-

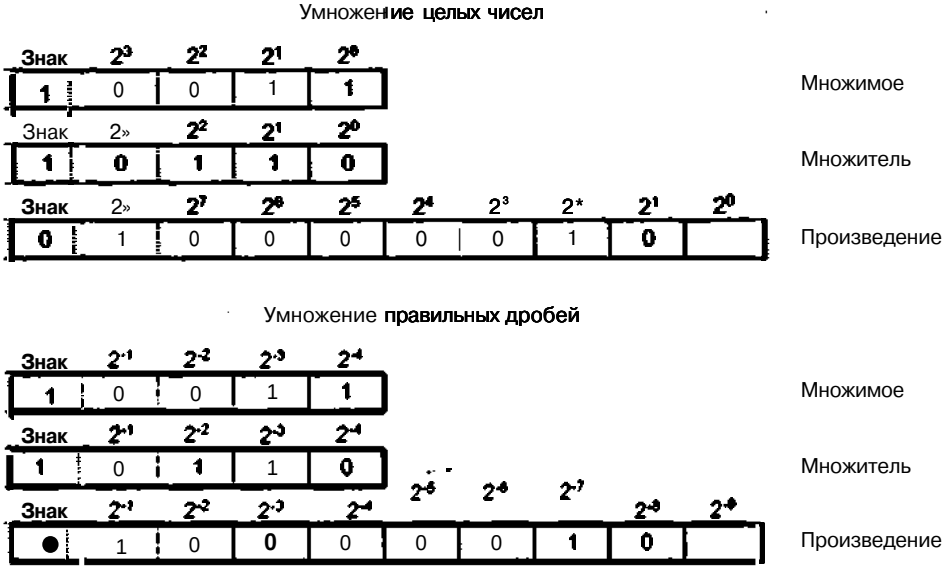


Рис. 7.23. Особенности представления произведения в разрядной сетке слова удвоенной разрядности

ны (2n разрядов). Поскольку число итераций в операции умножения определяется количеством цифровых разрядов множителя, окончательный результат может размещаться в разрядной сетке двойного слова неверно, что и имеет место при перемнбжении целых чисел (рис. 7.23). Как видно, младший разряд произведения целых чисел, имеющий вес 2^{0}, размещается в позиции двойного слова, соответствующей весу 2^{1}. Таким образом, для правильного расположения произведения в разрядной сетке двойного слова необходим дополнительный сдвиг вправо. Такой сдвиг можно учесть как в аппаратуре умножителя, так и программным способом.

С другой стороны, при перемножении правильных дробей дополнительный сдвиг не нужен (см. рис. 7.23). Это обстоятельство необходимо принимать во внимание при построении умножителя для чисел в форме с плавающей запятой, где участвующие в операции мантиссы представлены в нормализованном виде, то есть правильными дробями.

При умножении правильных дробей часто ограничиваются результатом, имеющим одинарную длину. В этом случае может применяться либо отбрасывание лишних разрядов, либо округление.

Ускорение целочисленного умножения

Методы ускорения умножения можно условно разделить на аппаратные и логические. Те и другие требуют дополнительных затрат оборудования, которые при использовании аппаратных методов возрастают с увеличением разрядности сомножителей. Аппаратные способы приводят к усложнению схемы умножителя, но не затрагивают схемы управления. Дополнительные затраты оборудования при реализации логических методов не зависят от разрядности операндов, но схема управления умножителя при этом утяжеляется. На практике ускорение умножения часто достигается комбинацией аппаратных и логических методов.

Логические методы ускорения умножения

Логические подходы к убыстрению умножения можно подразделить на две группы:

- методы, позволяющие уменьшить количество сложений в ходе умножения;
- методы, обеспечивающие обработку нескольких разрядов множителя за шаг.

Реализация и тех и других требует введения дополнительных цепей сдвига в регистры.

Рассмотрим первую группу логических методов.

Алгоритм Бута

В основе алгоритма Бута [63] лежит следующее соотношение, характерное для последовательностей двоичных цифр;

$$2m + 2^{m-1} + \ldots + 2^{k} = 2^{m+1} - 2^{k},$$

где m и k — номера крайних разрядов в группе из последовательных единиц. Например, $011110 - 2^{5} - 2^{1}$. Это означает, что при наличии в множителе групп из нескольких единиц (комбинаций вида 011,110), последовательное добавление к СЧП

множимого с нарастающим весом (от 2^k до 2^m) можно заменить вычитанием из СЧП множимого с весом 2^* и прибавлением к СЧП множимого с весом $2^{n+|}$.

Как видно, алгоритм предполагает три операции: сдвиг, сложение и вычитание. Помимо сокращения числа сложений (вычитаний) у него есть еще одно достоинство - он в равной степени применим к числам без знака и со знаком.

Алгоритм Бута сводится к перекодированию множителя из системы (0, 1) в избыточную систему (-1,0,1), из-за чего его часто называют перекодированием Бута (Booth recoding). В записи множителя в новой системе 1 означает добавление множимого к сумме частичных произведений, -1 - вычитание множимого и О не предполагает никаких действий. Во всех случаях после очередной итерации производится сдвиг множимого влево или суммы частичных произведений вправо. Реализация алгоритма предполагает последовательный в направлении справа налево анализ пар разрядов множителя — текущего b_i, и предшествующего $b_i(b_ib_{i-1})$. Для младшего разряда множителя (i - 0) считается, что предшествующий разряд равен 0, то есть имеет место пара b_00. На каждом шаге $i(i = 0,1,.,., n - 1)$ анализируется текущая комбинация $b_i(b_ib_{i-1})$.

Комбинация 10 означает начало цепочки последовательных единиц, и в этом случае производится вычитание множимого из СЧП.

Комбинация 01 соответствует завершению цепочки единиц, и здесь множимое прибавляется к СЧП.

Комбинация 00 свидетельствует об отсутствии цепочки единиц, а 11 — о нахождении внутри такой цепочки. В обоих случаях никакие арифметические операции не производятся.

По завершении описанных действий осуществляется сдвиг множимого влево либо суммы частичных произведений вправо, и цикл повторяется для следующей пары разрядов множителя.

Описанную процедуру рассмотрим на примерах (используется вариант со сдвигом множимого влево). В приведенных примерах операция вычитания, как это принято в реальных умножителях, выполняется путем сложения со множителем, взятым с противоположным знаком и представленным в дополнительном коде. Напомним, что для удлинения кода до нужного числа разрядов в дополнительные позиции слева заносится значение знакового разряда.

Пример 1.0110x0011-00010010 (в десятичном виде 6x3-18), После перекодирования Бута множитель (0,0,1,1) приобретает вид (0,1,0,-1).

В начале сумма частичных произведений принимается равной нулю - 00000000. Полагается, что младшему разряду множителя предшествовал 0. Дальнейший процесс поясняет рис. 7.24.

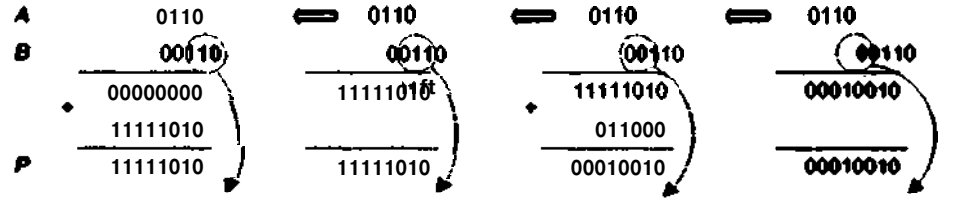


Рис. 7.24. Пример 1 умножения (6 x 3) в соответствии с алгоритмом Бута

Пример 2. 1100x0011 = 11110100 (в десятичной записи -4x3 --12). Процесс вычисления иллюстрирует рис. 7.25.

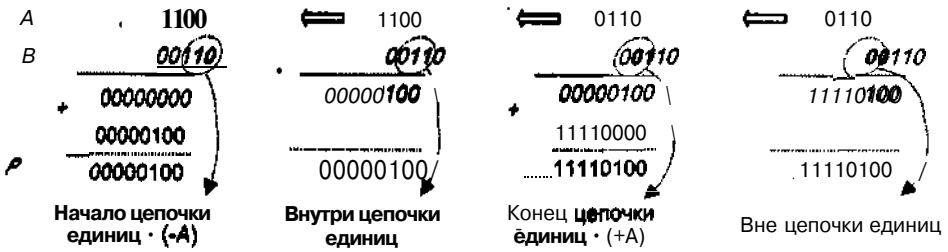


Рис. 7.25. Пример 2 умножения (-4 x 3) в соответствии с алгоритмом Бута

При наиболее благоприятном сочетании цифр множителя количество суммирований равно n/2, где *n* — число разрядов множителя.

Модифицированный алгоритм Бута

На практике большее распространение получила модификация алгоритма Бута, где количество операций сложения при любом сочетании единиц и нулей в множителе всегда равно *n/2.* В модифицированном алгоритме производится перекодировка цифр множителя из стандартной двоичной системы (0,1) в избыточную систему (-2, -1, 0, 1, 2), где каждое число представляет собой коэффициент, на который умножается множимое перед добавлением к СЧП. Одновременно анализируются три разряда множителя $b_i(b_ib_{i-1})$(два текущих и старший разряд из предыдущей тройки) и, в зависимости от комбинации 0 и 1 в этих разрядах, выполняется прибавление или вычитание множимого, прибавление или вычитание удвоенного множимого, либо никакие действия не производятся (табл. 7.1).

Таблица 7 . 1 . Логика модифицированного алгоритма Бута

x_i+1	x_i	x_i-1	Код (--2 -1 0 12)	Выполняемые действия
0	0	0	0	Не выполнять никаких действий
0	0	i	1	Прибавить к СЧП множимое
0	1	0	1	Прибавить к СЧП множимое
0	1	1	*1*	Прибавить к СЧП удвоенное множимое
i	0	0	-2	Вычесть из СЧП удвоенное множимое
1	0	1	-1	Вычесть из СЧП удвоенное множимое
1	1	0	-1	Вычесть из СЧП удвоенное множимое
1	1	1	0	Не выполнять никаких действий

Пример вычисления произведения 011001 x 101110 = 011000111110 (в десятичном виде 25 x (-18) = -450) показан на рис. 7.26.

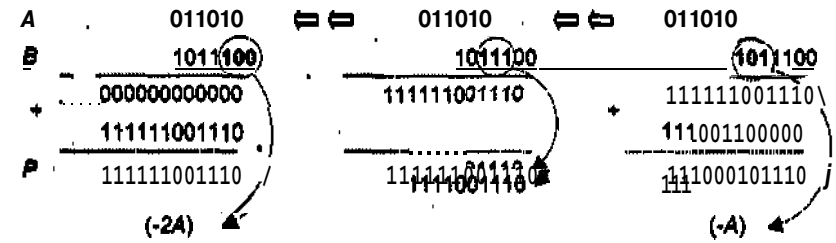


Рис. 7.26. Пример умножения (18 x (-25) в соответствии с модифицированным алгоритмом Бута

Алгоритм Лемана

Еще большее сокращение количества сложений может дать модификация, предложенная Леманом [151]. Здесь, даже при наименее благоприятном сочетании цифр множителя, количество операции сложения не превышает величины n/2, а в среднем же оно составляет я/3. Суть модификации заключается в следующем:

- если две группы нулей разделены единицей, стоящей в *к-й* позиции, то вместо вычитания в *к-й* позиции и сложения в *(к* +1)-й позиции достаточно выполнить только сложение в k-й позиции;
- если две группы единиц разделены нулем, стоящим в k-й позиции, то вместо сложения в k-й позиции и вычитания в *(к* + 1)-й позиции достаточно выполнить только вычитание в *к-й* позиции.

Обработка двух разрядов множителя за шаг

Из второй группы логических методов остановимся на умножении с обработкой за шаг двух разрядов множителя (IBM 360/370). В принципе это более эффективная версия алгоритма Бута. Анализ множителя начинается с младших разрядов. В зависимости от входящей двухразрядной комбинации предусматриваются следующие действия:

- 00 — простой сдвиг на два разряда вправо суммы частичных произведений (СЧП);
- 01 — к СЧП прибавляется одинарное множимое, после чего СЧП сдвигается на 2 разряда вправо;
- 10 — к СЧП прибавляется удвоенное множимое, и СЧП сдвигается на 2 разряда вправо;
- 11 — из СЧП вычитается одинарное множимое, и СЧП сдвигается на 2 разряда вправо. Полученный результат должен быть скорректирован на следующем шаге, что фиксируется в специальном триггере признака коррекции.

Так как в случае пары 11 из СЧП вычитается одинарное множимое вместо прибавления утроенного, для корректировки результата к СЧП перёд выполнением сдвига надо было бы прибавить учетверенное множимое. Но после сдвига на два разряда вправо СЧП уменьшается в четыре раза, так что на следующем шаге достаточно добавить одинарное множимое. Это учитывается при обработке следую-

щей пары разрядов множителя, путем обработки пары 00 как 01, пары 01 как 10, 10 —как 11, а 11 —как 00. В последних двух случаях фиксируется признак коррекции.

Таблица 7.2. Формирование признака коррекции

Пара разрядов множителя	+1 из предыдущей пары	+1 в следующую пару	Знак действия	Кратность множимому
00	0	0		0
01	0	0	+	1
to	0	0	+	2
11	0	1	-	1
00	t	0	+	t
01	1	0	+	2
10	1	1	-	1
11	1	1		0

Правила обработки пар разрядов множителя с учетом признака коррекции приведены в табл. 7.2. После обработки каждой комбинации содержимое регистра множителя и сумматора частичных произведений сдвигается на 2 разряда вправо. Данный метод умножения требует корректировки результата, если старшая пара разрядов множителя равна 11 или 10 и состояние признака коррекции единичное. В этом случае к полученному произведению должно быть добавлено множимое.

Аппаратные методы ускорения умножения

Традиционный метод умножения за счет сдвигов и сложений, даже при его аппаратной реализации, не позволяет достичь высокой скорости выполнения операции умножения. Связано это, главным образом, с тем, что при добавлении к СЧП очередного частичного произведения перенос должен распространиться от младшего разряда СЧП к старшему. Задержка из-за распространения переноса относительно велика, причем она повторяется при добавлении каждого ЧП.

Один из способов ускорения умножения состоит в изменении системы кодирования сомножителей, за счет чего можно сократить количество суммируемых частичных произведений. Примером такого подхода может служить алгоритм Бута.

Еще один ресурс повышения производительности умножителя — использование более эффективных способов суммирования ЧП, исключающих затраты времени на распространение переносов. Достигается это за счет представления ЧП в избыточной форме, благодаря чему суммирование двух чисел не связано с распространением переноса вдоль всех разрядов числа. Наиболее употребительной формой такого избыточного кодирования является так называемая форма *с сохранением переноса*. В ней каждый разряд числа представляется двумя битами *cs,* известными как перенос (с) и сумма *(s)*. При суммировании двух чисел в форме с сохранением переноса перенос распространяется не далее, чем на один разряд.

Это делает процесс суммирования значительно более быстрым, чем в случае сложения с распространением переноса вдоль всех разрядов числа.

Наконец, третья возможность ускорения операции умножения заключается в параллельном вычислении всех частичных произведений. Если рассмотреть общую схему умножения (рис. 7.27), то нетрудно заметить, что отдельные разряды ЧП представляют собой произведения вида $a_i b_j$, то есть произведение определенного бита множимого на определенный бит множителя. Это позволяет вычислить все' биты частичных произведений одновременно, с помощью n^2 схем «И». При перемножении чисел в дополнительном коде отдельные разряды ЧП могут иметь вид $a_i\overline{b_j}$, $\overline{a}ib$ или $\overline{a_i b_j}$. Тогда элементы "И"заменяются элементами, реализующими соответствующую логическую функцию.

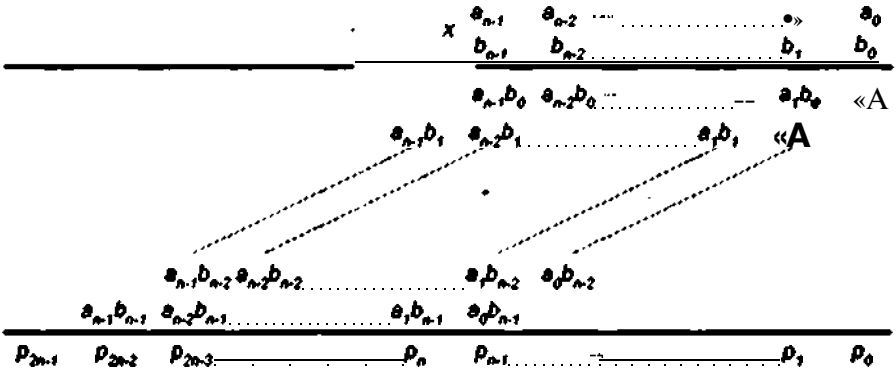


Рис. 7.27. Схема перемножения п-разрядных чисел без знака

Таким образом, аппаратные методы ускорения умножения сводятся:

- *к* параллельному вычислению частичных произведений;
- к сокращению количества операций сложения;
- к уменьшению времени распространения переносов при суммировании частичных произведений.

Все три подхода в любом их сочетании обычно реализуются с помощью комбинационных устройств.

Параллельное вычисление ЧП имеет место практически во всех рассматриваемых ниже схемах умножения. Различия проявляются в основном в способе суммирования полученных частичных произведений, и с этих позиций используемые схемы умножения можно подразделить на *матричные* и с *древовидной структурой*. В обоих вариантах суммирование осуществляется с помощью массива взаимосвязанных одноразрядных сумматоров. В матричных умножителях сумматоры организованы в виде матрицы, а в древовидных они реализуются в виде дерева того или иного типа.

Различия в рамках каждой из этих групп выражаются в количестве используемых сумматоров, их виде и способе распространения переносов, возникающих в процессе суммирования.

В *матричных умножителях* суммирование осуществляется матрицей сумматоров, состоящей из последовательных линеек (строк) одноразрядных сумматоров с сохранением переноса (ССП). По мере движения данных вниз по массиву сумматоров каждая строка ССП добавляет к СЧП очередное частичное произведение. Поскольку промежуточные СЧП представлены в избыточной форме с сохранением переноса, во всех схемах, вплоть до последней строки, где формируется окончательный результат, распространения переноса не происходит. Это означает, что задержка в умножителях отталкивается только от «глубины» массива (числа строк сумматоров) и не зависит от разрядности операндов, если только в последней строке матрицы, где формируется окончательная СЧП, не используется схема с последовательным переносом.

Наряду с высоким быстродействием важным достоинством матричных умножителей является их регулярность, что особенно существенно при реализации таких умножителей в виде интегральной микросхемы. С другой стороны, подобные схемы занимают большую площадь на кристалле микросхемы, причем с увеличением разрядности сомножителей эта площадь увеличивается пропорционально квадрату числа разрядов. Вторая проблема с матричными умножителями — низкий уровень утилизации аппаратуры. По мере движения СЧП вниз каждая строка задействуется лишь однократно, когда ее пересекает активный фронт вычислений. Это обстоятельство, однако, может быть затребовано для повышения эффективности вычислений путем конвейеризации процесса умножения, при которой по мере освобождения строки сумматоров последняя может быть использована для умножения очередной пары чисел:

Ниже рассматриваются различные алгоритмы умножения и соответствующие им схемы матричных умножителей. Каждый из алгоритмов имеет свои плюсы и минусы, важность которых для пользователя определяет выбор той или иной схемы.

Матричное умножение чисел без знака

Результат *P* перемножения двух «-разрядных двоичных целых чисел *A и B* без знака можно описать выражением

$$P = A \times B = \left(\sum_{i=0}^{n-1} a_i \times 2^i\right) \times \left(\sum_{j=0}^{n-1} b_j \times 2^j\right) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_i b_j \times 2^{i+j}.$$

Умножение сводится к параллельному формированию битов из *n n*-разрядных частичных произведений с последующим их суммированием с помощью матрицы сумматоров, структура которой соответствует приведенной матрице умножения. Схема известна как *умножитель Брауна*. На рис. 7.28 показан такой умножитель для четырехразрядных двоичных чисел в котором каждому столбцу в матрице умножения соответствует диагональ умножителя. Биты частичных произведений (ЧП) вида *aibj* формируются с помощью элементов «И». Для суммирования ЧП применяются два вида одноразрядных сумматоров с сохранением переноса: полусумматоры (ПС)[1] и полные сумматоры (СМ)[2].

[1] Полусумматором называется одноразрядное суммирующее устройство, имеющее два входа для слагаемых и два выхода — выход бита суммы к выход бита переноса.

[2] В отличие от полусумматора складывает три числа, то есть имеет три входа для слагаемых и два выхода — выход бита суммы и выход бита переноса.

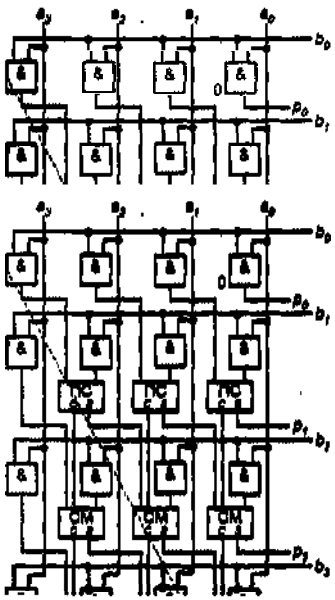


Рис. 7.2а. Матричный умножитель Брауна для четырехразрядных чисел без знака

Матричный умножитель $n \times n$ содержит n^2 схем «И», n ПС и $(n^2 - 2n)$ СМ. Если принять, что для реализации полусумматора требуются два логических элемента, а для полного сумматора - пять, то общее количество логических элементов в умножителе составляет $n^2 + 2n$ или $n^2 - 2n$, или $6n$ или $8n$.

Быстродействие умножителя определяется наиболее длинным маршрутом распространения сигнала, который в худшем случае (пунктирная линия на рис. 7.28) включает в себя прохождение одной схемы «И», двух ПС и (2n - 4) СМ. Полагая задержки в схеме «И» и полусумматоре равными Δ, а в полном сумматоре — 2Δ, общую задержку в умножителе можно оценить выражением $(4n - 5)\Delta$. Чтобы сократить ее длительность, n-разрядный сумматор с последовательным переносом в нижней, строке умножителя можно заменить более быстрым вариантом сумматора. Последнее, однако, не всегда желательно, поскольку это увеличивает число используемых в умножителе логических элементов и ухудшает регулярность схемы.

В общем случае задержка в матричных умножителях пропорциональна их разрядности: O(n).

Матричное умножение чисел в дополнительном коде

К сожалению, умножитель Брауна годится только для перемножения чисел без знака. При обработке знаковых чисел отрицательные представляются дополнительным кодом, а матричные умножителя строятся по схемам, отличным от схемы Брауна. Прежде всего, напомним, что запись двоичного числа в дополнительном коде (с дополнением до 2) имеет вид

$$X = x_{n-1}x_{n-2} \ldots x_1x_0 = -x_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} x_i \times 2^i,$$

где первый член правого выражения представляет знак числа, а сумма — его модуль.

Исходя из приведенной записи, произведение *Р двух* «-разрядных двоичных целых чисел *А и В* дополнительном коде (значение произведения и сомножителей в дополнительном коде обозначим соответственно *V(P), V(A)* и *V(B))* можно описать выражением

$$V(P) = V(A) \times V(B) = (-a_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} a_i \times 2^i) \times (-b_{n-1} \cdot 2^{n-1} + \sum_{j=0}^{n-2} b_j \times 2^j) =$$

$$= a_{n-1}b_{n-1} \times 2^{2n-2} + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} a_i b_j 2^{i+j} - \sum_{i=0}^{n-2} a_i b_{n-1} \times 2^{i+n-1} - \sum_{j=0}^{n-2} a_{n-1} b_j \times 2^{j+n-1} =$$

$$= a_{n-1}b_{n-1} \times 2^{2n-2} + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} a_i b_j \times 2^{i+j} + \sum_{i=0}^{n-2} \overline{a_i b_{n-1}} \times 2^{i+n-1} + \sum_{j=0}^{n-2} \overline{a_{n-1} b_j} \times 2^{j+n-1} +$$

$$+ 2^n + 2^{2n-1}.$$

Матрица умножения чисел со знаком, представленных в дополнительном коде, похожа на матрицу перемножения чисел без знаков (рис. 7.29). Отличие состоит в том, что (2n - 2) частичных произведений инвертированы, а в столбцы *n* и (2n - 1) добавлены единицы.

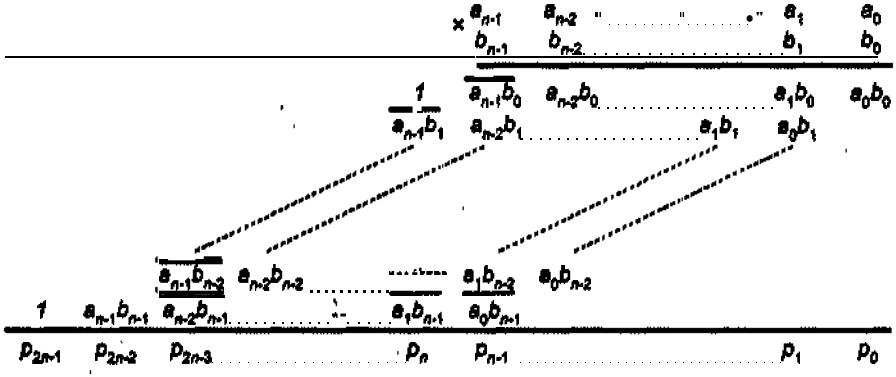


Рис. 7.29. Матрица перемножения п-разрядных чисел в дополнительном коде

Соответствующая схема матричного умножителя для четырехразрядных чисел показана на рис. 7.30.

Здесь (2n - 2) частичных произведений инвертированы за счет замены элементов «И» на элементы «И-НЕ». Сумматор в младшем разряде нижнего ряда складывает 1 в столбце *n* с вектором сумм и переносов из предшествующей строки, реализуя при этом следующие выражения:

$$s_i = \overline{a_i \oplus b_i}; \quad c_{i+1} = a \times b_i.$$

Инвертор в нижней строке слева обеспечивает добавление единицы в столбец (2n-1).

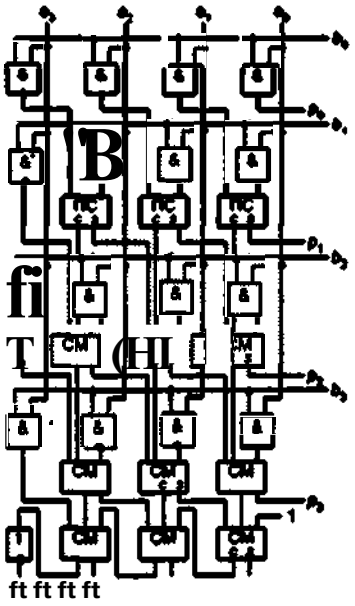


Рис. 7.30. Матричный умножитель для четырехразрядных чисел в дополнительном коде

Алгоритм Бо-Вули

Несколько иная схема матричного умножителя, также обеспечивающего умножение чисел в дополнительном коде, была предложена Бо и Вули [61]. В алгоритме Бо-Вули произведение чисел в дополнительном коде представляется следующим соотношением:

$$V(P) = V(A) \times V(B) = \left(-a_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} a_i \times 2^i\right) \times \left(-b_{n-1} \times 2^{n-1} + \sum_{j=0}^{n-2} b_j \times 2^j\right) =$$

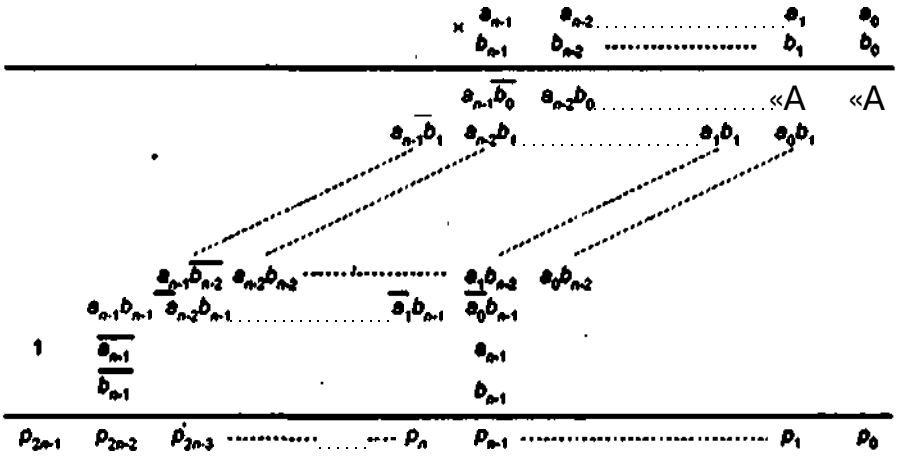
$$= a_{n-1} b_{n-1} \times 2^{2n-2} + \sum_{i=0}^{n-2} \sum_{j=0}^{n-2} a_i b_j \times 2^{i+j} - \sum_{i=0}^{n-2} a_i b_{n-1} \times 2^{i+n-1} - \sum_{j=0}^{n-2} a_{n-1} b_j \times 2^{j+n-1} =$$

$$= -2^{2n-1} + (\overline{a_{n-1}} + \overline{b_{n-1}} + a_{n-1} b_{n-1}) \times 2^{2n-2} + \sum_{i=0}^{n-2} \sum_{j=0}^{n-2} a_i b_j \times 2^{i+j} + \sum_{i=0}^{n-2} \overline{a_i} b_{n-1} \times 2^{i+n-1} +$$

$$+ \sum_{j=0}^{n-2} a_{n-1} \overline{b_j} \times 2^{j+n-1} + (a_{n-1} + b_{n-1}) \times 2^{n-1}.$$

Матрица умножения, реализующая алгоритм, приведена на рис. 7.31, а соответствующая ей схема умножителя — на рис. 7.32.

По ходу умножения частичные произведения, имеющие знак «минус», смещаются к последней ступени суммирования. Недостатком схемы можно считать то, что на последних этапах работы требуются дополнительные сумматоры, из-за чего регулярность схемы нарушается.



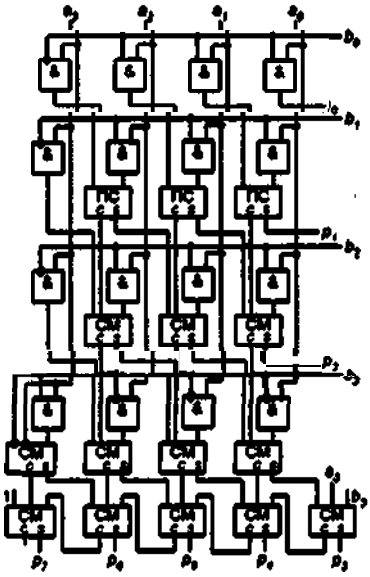


Рис. 7 . 3 1 . Матри о алгоритму Бо-Вули

Рис. 7.32. Матричный умножитель для четырехразрядных чисел в дополнительном коде по схеме Бо-Вули

Алгоритм Пезариса

Еще один алгоритм для вычислений произведения чисел в дополнительном коде был предложен Пезарисом [181].

При представлении числа в дополнительном коде старший разряд числа имеет отрицательный вес. Для учета этого обстоятельства Пезарис выдвигает идею использовать в умножителе четыре вида полных сумматоров (рис, 7.33).

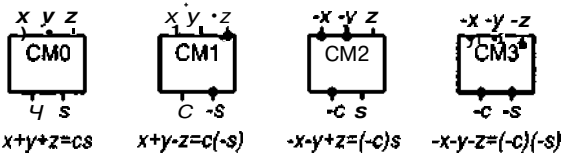


Рис. 7.33. Виды сумматоров, применяемых в матричном умножителе Пезариса

В сумматоре типа СМО, который фактически является обычным полным сумматором, все входные данные *(x, y, z)* имеют положительный вес, а результат лежит в диапазоне 0-3. Этот результат представлен двухразрядным двоичным числом *cs,* где *с* и s также присвоены положительные веса. В остальных трех типах сумматоров некоторые из сигналов имеют отрицательный вес. Схема умножения в рассматриваемом методе показана на рис. 7.34.

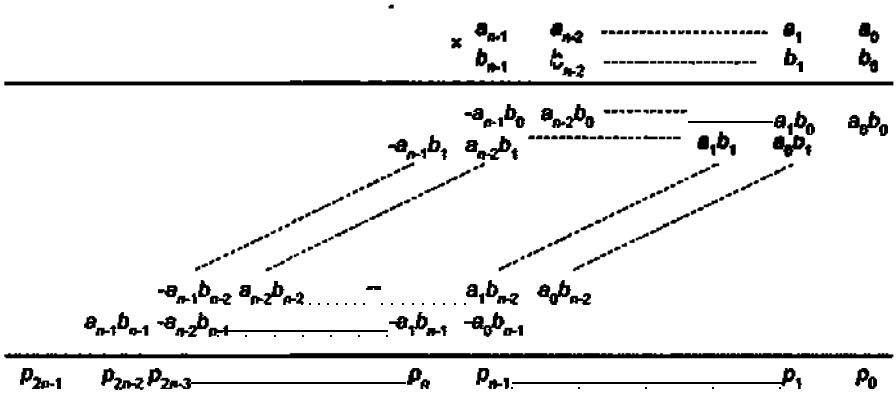


Рис. 7.34. Матрица перемножения n-разрядных чисел согласно алгоритму Пезариса

Здесь знак "минус"- трактуется следующим образом: -1 = -2 * 1 + 1; -0 = -2 * *0 + 0. Схема умножителя, реализующего алгоритм Пезариса, приведена на рис. 7.35,

По сравнению с умножителем Бо-Вули, схема Пезариса имеет более регулярный вид, но, с другой стороны, она предполагает присутствие нескольких типов сумматоров.

Древовидные умножители

Сократить задержку, свойственную матричным умножителям, удается в схемах, построенных по древовидной структуре. Если в матричных умножителях для суммирования *n* частичных произведений требуется « строк сумматоров, то в древовидных схемах количество ступеней сумматоров пропорционально $\log_2$ n(рис. 7.36).

Соответственно числу ступеней суммирования сокращается и время вычисления СЧП. Хотя древовидные схемы быстрее матричных, однако при их реализации требуются дополнительные связи для объединения разрядов, имеющих одинаковый вес, из-за чего площадь, занимаемая схемой на кристалле микросхемы, может оказаться даже больше, чем в случае матричной организации сумматоров. Еще одна проблема связана с тем, что стандартное двоичное дерево не является самой эффективной древовидной иерархией, поскольку не позволяет в полной мере

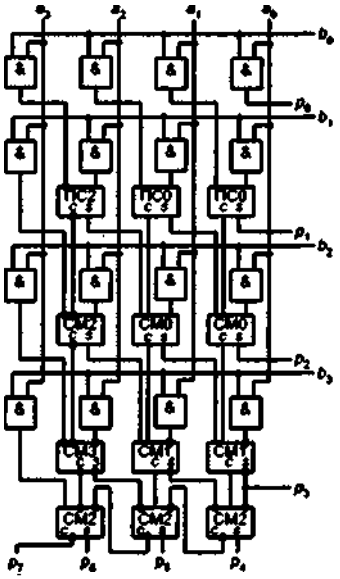


Рис. 7.35. Матричный умножитель для четырехразрядных чисел в дополнительном коде по схеме Пеэариса

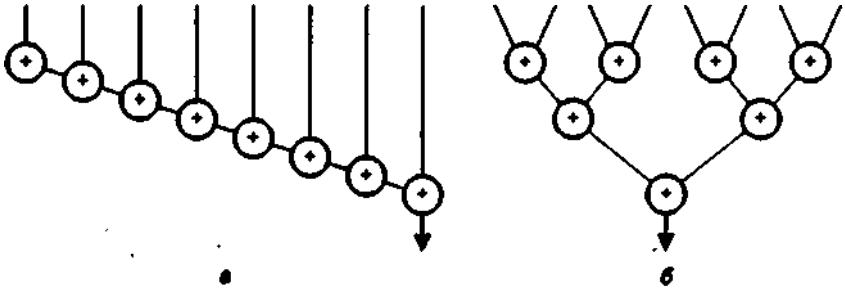


Рис. 7.36. Суммирование частичных произведений в умножителях: а — о матричной структурой; *б* — со структурой двоичного дерева

воспользоваться возможностями полного сумматора (имеющего не два, а три входа), благодаря чему можно одновременно суммировать сразу три входных бита. По этой причине на практике в умножителях с древовидной структурой применяют иные древовидные схемы. С другой стороны, такие схемы не столь регулярны, как двоичное дерево, а регулярность структуры — одно из основных требований при создании интегральных микросхем.

Древовидные умножители включают в себя три ступени:

- ступень формирования битов частичных произведений, состоящую из n^2 элементов «И»;
- ступень сжатия частичных произведений — реализуется в виде дерева параллельных сумматоров (накопителей), служащего для сведения частичных произведений к вектору сумм и вектору переносов. Сжатие реализуется несколь-

кими рядами сумматоров, причем каждый ряд вносит задержку, свойственную одному полному сумматору;

- ступень заключительного суммирования, где осуществляется сложение вектора сумм й вектора переносов с целью получения конечного результата. Обычно здесь применяется быстрый сумматор с временем задержки, пропорциональным $0(\log_2(n))$.

Известные древовидные умножители различаются по способу сокращения ч пела ЧП. При использовании в умножителе СМ и ПС их обычно называют счетчиками (3,2) и (2,2) соответственно. Связано это с тем, что код на выходах *cs,* как и в двоичном счетчике, равен количеству единиц, поданных на входы.

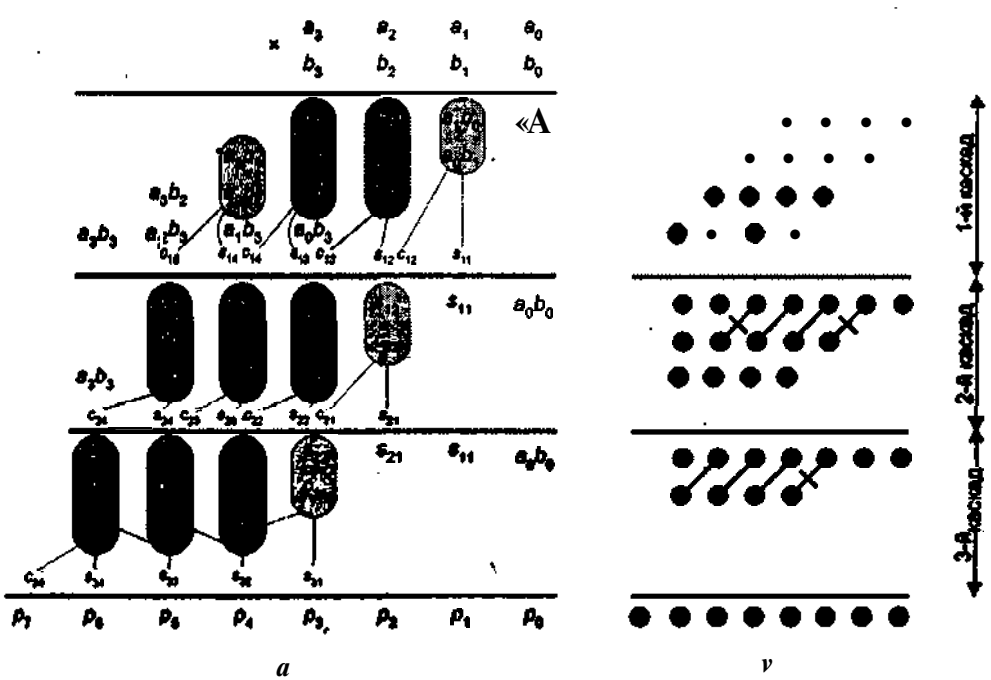
Процесс «компрессии» СЧП завершается формированием двух векторов - вектора сумм и вектора переносов, которые для получения окончательного результата обрабатываются многоразрядными сумматорами, то есть различие между древовидными схемами сжатия касается, главным образом, способа формирования упомянутых векторов.

В известных на сегодня умножителях наибольшее распространение получили три древовидных схемы суммирования ЧП: дерево Уоллеса, дерево Дадда и перевернутое ступенчатое дерево.

В наиболее общей формулировке дерево Уоллеса - это оператор с *n* входами и $\log_2 n$ выходами, в котором код на выходе равен числу единиц во входном коде. Вес битов на входе совпадает с весом младшего разряда выходного кода. Простейшим деревом Уоллеса является СМ. Используя такие сумматоры, а также полусумматоры, можно построить дерево Уоллеса для перемножения чисел любой разрядности, при этом количество сумматоров возрастает пропорционально величине $\log_2 n$. В такой же пропорции растет время выполнения операции умножения.

Согласно алгоритму Уоллеса, строки матрицы частичных произведений группируются по три. Полные сумматоры используются для сжатия столбцов стремя битами, а полусумматоры - столбцов с двумя битами. Строки, не попавшие в набор из трех строк, учитываются в следующем каскаде редукции. Количество строк в матрице (ее высота) на j-й ступени определяется выражениями $w_0 = n$ и $w_{j+1} = 2\lfloor w_j/3 \rfloor + (V \bmod 3)$, пока $w_j >= 2$

В 32-разрядном умножителе на базе дерева Уоллеса высоты матриц ЧП последовательно уменьшаются в последовательности: 22,15, 10, 7, 5, 4, 3 и 2. Логика построения дерева Уоллеса для суммирования частичных произведений в умно жителе 4x4 показана на рис, 7.37, *а*. Для пояснения структуры дерева сумматоров часто применяют так называемую точечную диаграмму (рис. 7.37,*б*}. В ней точки обозначают биты частичных произведений, прямые диагональные линии представляют выходы полных сумматоров, а перечеркнутые диагонали — выходы полусумматоров. Хотя на рис. 7.37, *а* в третьем каскаде показаны три строки, фактически после редукции остаются лишь две первых, а третья лишь отражает переносы, которые учитываются при окончательном суммировании. Этим объясняется кажущееся отличие от точечной диаграммы.



а *б*

Рис. 7.37. Суммирование ЧП с помощью дерева Уоллеса (вариант 1): *а* - логика суммирования; б — точечная диаграмма

Умножитель (рис. 7.38) состоит из трех ступеней с высотами 4,3,2 и содержит 16 схем «И», 3 полусумматора, 5 полных сумматоров. Сложение векторов сумм и переносов в последнем каскаде реализуется четырехразрядным сумматором с последовательным распространением переноса, однако чаще для ускорения привлекаются более эффективные схемы распространения переноса, например параллельная.

Отметим, что избыточность кодирования, заложенная в алгоритм Уоллеса, приводит к тому, что возможно построение различных вариантов схемы дерева. Так, на рис. 7.39 показана иная реализация дерева Уоллеса для четырехразрядных операндов.

Как видно, новый вариант схемы никакого выигрыша в аппаратурном плане не дает.

Схема Уоллеса считается наиболее быстрой, но в то же время ее структура наименее регулярна, из-за чего предпочтение отдается иным древовидным структурам. Основная сфера использования умножителей со схемой Уоллеса - перемножение чисел большой разрядности. В этом случае быстродействие имеет превалирующее значение.

При умножении чисел небольшой разрядности более распространена другая схема сжатия суммирования ЧП — схема дерева Л. Дадда. В ее основе также лежит дерево Уоллеса, но реализуемое минимальным числом сумматоров.

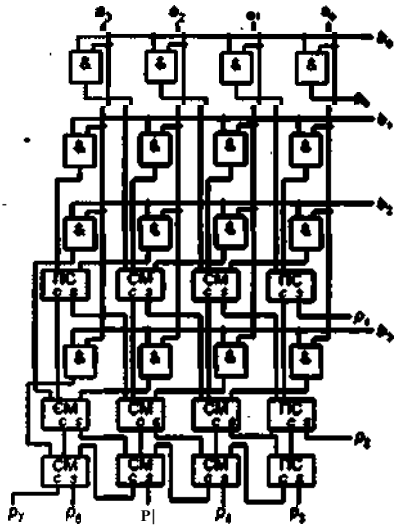


Рис. 7.38. Умножитель 4 x 4 со структурой дерева Уоллеса

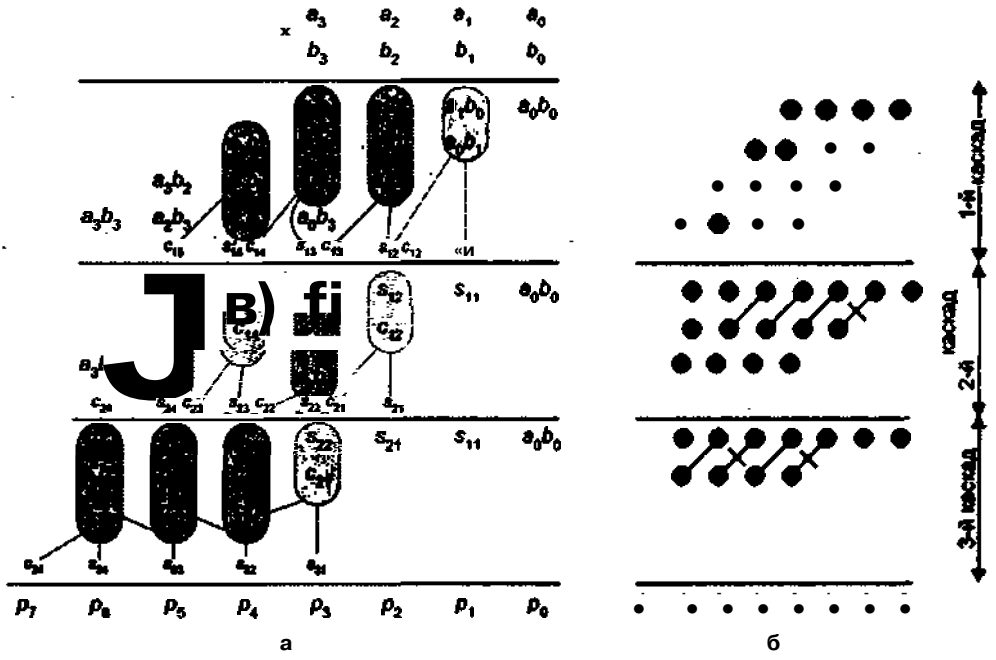


Рис. 7.39. Суммирование ЧП с помощью дерева Уоллеса (вариант 2): *а* - логика суммирования; *б* - точечная диаграмма

Схема редукции, предложенная Л. Даддом, начинается с определения высоты промежуточных матриц частичных произведений: d_1-2 и d_{j+l} =/1,5 *x dj J,* пока *dj* < *n*. Значения для d_j приведены в табл. 7.3.

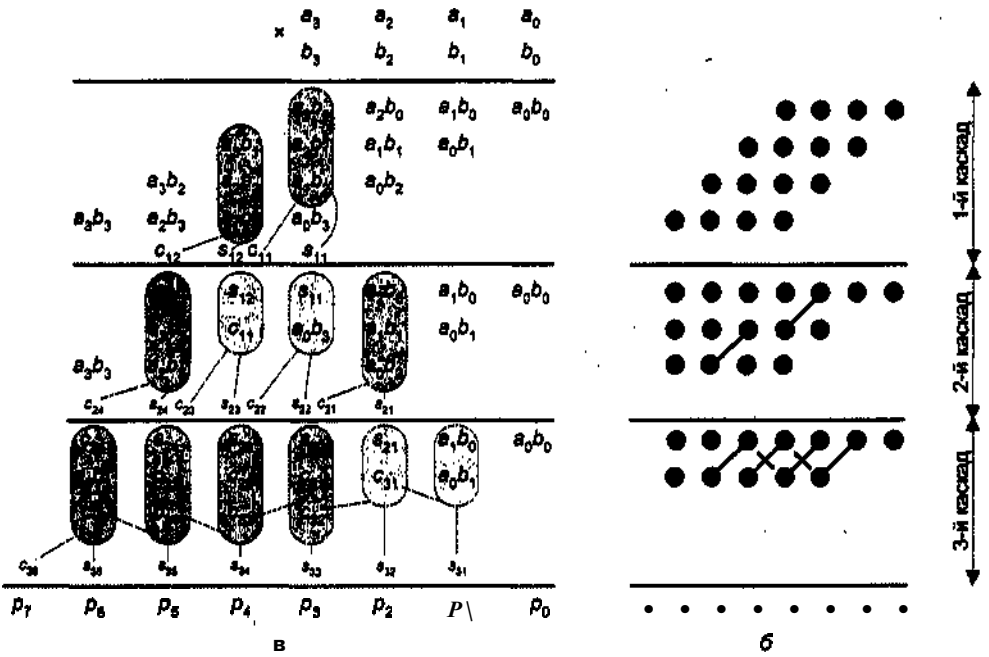
Таблица 7.3. Таблица высот промежуточных матриц в алгоритме Дадда

1	1	2	3	4	5	6	7	8	9	10	11
d_j	2	3	4	6	9	13	19	29	42	63	94

Так, 32-разрядный умножитель на базе дерева Дадда имеет высоты промежуточных матриц 29,19,13,9,6,4,3 и 2. На j-й от конца ступени умножителя используется минимальное число ПС и СМ, позволяющее сократить число битов в столбце d_j Если высота столбца, включая переносы, равна h, то число полусумматоров ($N_{ПС}$) и полных сумматоров ($N_{СМ}$) составляет:

$$N_{СМ} = \lfloor (h - d_j)/2 \rfloor; \quad N_{ПС} = (h - d_j) \bmod 2.$$

На рис. 7.40 описан умножитель 4x4, реализующий алгоритм дерева Дадда. Для этого требуется 16 схем "И", два полусумматора, четыре полных сумматора и шестиразрядный сумматор. Схема содержит три ступени с высотами промежуточных матриц: 4,3 и 2. На этапе суммирования вектора сумм и вектора переносов необходим (2n - 2)-разрядный сумматор.



Рис, 7.40. Суммирование ЧП с помощью дерева Дадда для случая чисел без знака: *а* - логика суммирования; *б* - точечная диаграмма

Схема умножения чисел в дополнительном коде, рассмотренная применительно к матричному умножителю, может быть адаптирована *и* для умножителя со схемой Дадда. В таком случае схема сжатия приобретает вид, показанный на рис. 7.41.

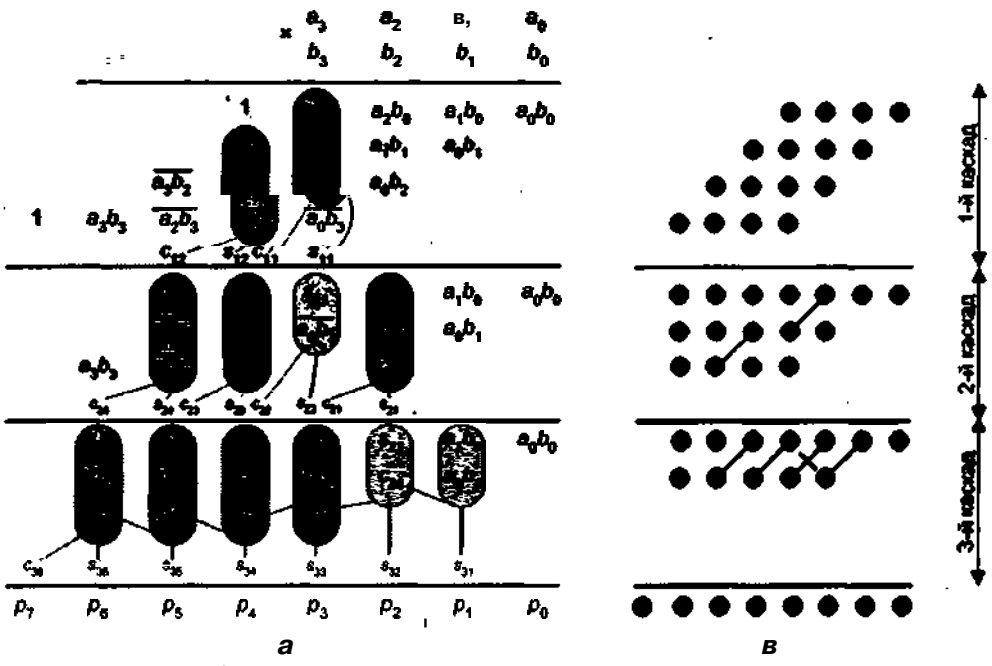


Рис. 7 . 4 1 . Суммирование ЧП в дополнительном коде с помощью дерева Дадда: *а* - логика суммирования; *б* — точечная диаграмма

Различия методов Уоллеса и Дадда являются следствием разных подходов к решению задачи "компрессии" суммирования. Алгоритм Уоллеса ориентирован на сжатие кодов как можно раньше, на самых ранних этапах, а алгоритм Дадда стремится это сделать по возможности позже, то есть наибольший уровень сжатия относит к завершающим стадиям.

Сравнивая схемы Уоллеса и Дадда, можно отметить, что число каскадов сжатия в них одинаково, однако количество используемых полусумматоров и полных сумматоров в схеме Дадда меньше (при подсчете числа элементов обычно не учитывают многоразрядные сумматоры, предназначенные для окончательного сложения векторов сумм и переносов). С другой стороны, на этапе сложения векторов сумм и переносов в варианте Уоллеса требуется сумматор с меньшим числом разрядов (в нашем примере — 4 против 6).

У обеих схем имеется общий недостаток — нерегулярность структуры, особенно у дерева Уоллеса.

Схема перевернутой лестницы (overturned stairs), предложенная в [169], является собой одну из попыток сделать древообразную структуру более регулярной, а значит, облегчить ее реализацию в интегральном исполнении. «Лестница»строится из базовых блоков трех видов (рис. 7.42, *а),* которые авторы назвали «ветвью» (branch), «соединителем» (connector) и «корнем» (root).

Базовые элементы объединяются, образуя дерево, имеющее *n* входов, Подобная схема на 18 входов показана на рис. 7.42,*б.* Как видно, дерево имеет достаточ-

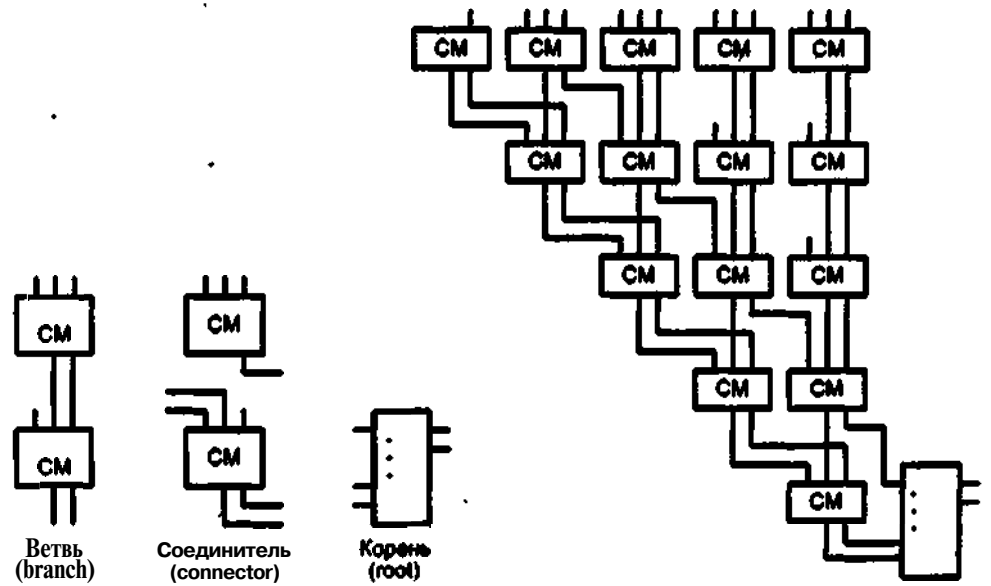


Рис. 7.42. Перевернутое дерево: ***а*** - базовые блоки; *б* — структура дерева на 18 входов

но регулярную структуру, однако нужно учитывать, что веса отдельных входов различаются. Кроме того, конструкция умножителя получается сложной.

Сравнительная оценка схем умножения с матричной и древообразной структурой

В табл. 7.4 приведены данные по производительности различных видов умножителей, выполненных средствами интегральной схемотехники. Быстродействие умножителей характеризуется коэффициентом при величине задержки T_L В ОДНОМ логическом элементе.

Алгоритм	Задержка при τ_L	Количество транзисторов
Алгоритм	Задержка при τ_L	Количество транзисторов
Брауна	$64n$ - 72,4	$32n^2 - 36n$
Дадда 8 × 8	280,2	1760
Дадда 16 × 16	544,2	7616
Пезариса	$46,4n - 38,4$	$32n^2 - 26n$
Бо~Вули	$46,4n - 14,4$	$32n^2 - 32n + 88$
Бута	$41,5n - 1,9$	$23n^2 + 5n - 58$

[1] n - разрядность сомножителей.

Содержимое таблицы плюс некоторые не включенные в нее данные позволяют сделать следующие выводы. Наиболее быстро работают умножители, построенные по схеме Бута, а также имеющие древовидную структуру, в частности дерево Дадда. Для операндов длиной в 16 разрядов и более наиболее привлекательной представляется модифицированная схема Бута, как по скорости, так и по затратам оборудования. Максимально" быстрое выполнение операции умножения обеспечивает сочетание алгоритма Бута и дерева Уоллеса. С другой стороны, достаточно хорошие показатели скорости при умножении чисел небольшой разрядности выдает схема Бо-Вули. В плане потребляемой мощности наиболее экономичными являются умножители, построенные по схемам Брауна и Пезариса. Несмотря на сравнительно небольшое число используемых транзисторов, схемы на базе алгоритма Бута, а также древовидные реализации, потребляют больше из-за избыточных внутренних связей, связанных с нерегулярной структурой этих схем.

Конвейеризация параллельных умножителей

В матричной и древовидной структурах параллельных умножителей заложен еще один потенциал повышения производительности - возможность конвейеризации. При конвейеризации весь процесс вычислений разбивается на последовательность законченных шагов. Каждый из этапов процедуры умножения выполняется на своей ступени конвейера, причем все ступени работают параллельно. Результаты, полученные на i-й ступени, передаются на дальнейшую обработку в (i + 1)-ю ступень конвейера. Перенос информации со ступени на ступень происходит через буферную память, размещаемую между ними (рис. 7.43).

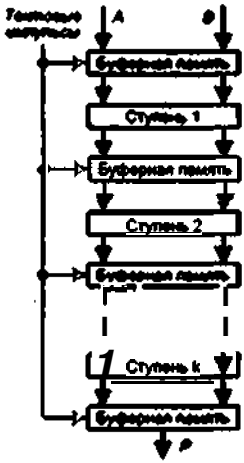


Рис. 7.43. Структура конвейерного умножителя

Выполнившая свою операцию ступень помещает результат в буферную память и может приступать к обработке следующей порции данных операций, в то время как очередная ступень конвейера в качестве исходных использует данные, хранящиеся в буферной памяти на ее входе. Синхронность работы конвейера обеспечивается тактовыми импульсами, период которых т определяется самой медленной

ступенью конвейера t_i и задержкой в элементе буферной памяти t_i $t = \max(t_1, t_2, \ldots t_k) + t_i$. Несмотря на то что время выполнения операции умножения для каждой конкретной пары сомножителей в конвейерном умножителе не только не уменьшается, но даже несколько увеличивается за счет задержек в буферной памяти при последовательном перемножении последовательностей пар сомножителей, достигаемый выигрыш весьма ощутим. Действительно, в конвейерном умножителе из k ступеней перемножаемые данные могут подаваться на вход с интервалом в k раз меньшим, чем в случае обычного умножителя. В том же темпе появляются и результаты на выходе.

Схема конвейера легко может быть применена к матричным и древовидным умножителям. В матричных умножителях в качестве ступени конвейера выступает каждая строка матрицы сумматоров. В качестве примера конвейеризированного матричного умножителя на рис. 7,44 приведена схема *4x4*. Черными прямоугольниками обозначены триггеры-защелки, образующие буферную память.

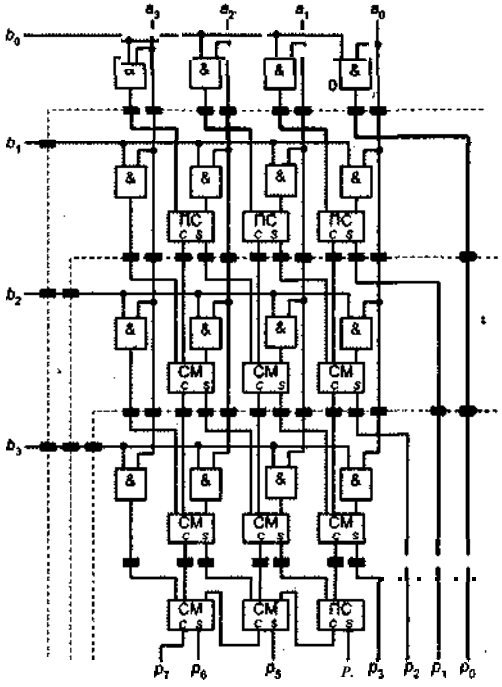


Рис. 7.44. Конвейеризированный матричный умножитель

Конвейеризация матричных умножителей на уровне строк сумматоров может быть затруднительной из-за большого числа ступеней и необходимости введения в состав умножителя значительного количества триггеров-защелок. Сокращение числа триггеров достигается за счет следующих приемов:

- отказа от использования идеи конвейеризации между входными схемами "И" и первой строкой полных сумматоров;

- увеличением времени обработки на каждой ступени, например можно принять его равным удвоенному времени срабатывания полного сумматора;
- отказом от формирования всех n^2 битов частичных произведений в самом начале, перед первой ступенью конвейера, и вычислением их по мере необходимости на разных ступенях конвейера.

В древовидных умножителях в качестве ступеней конвейера выступают каскады сжатия, то есть более крупные образования, чем в матричных умножителях, Кроме того, количество каскадов компрессии также значительно меньше. Это делает конвейеризацию древовидных умножителей более привлекательной. На рис. 7.45 показана точечная диаграмма конвейеризированного умножителя со схемой Дадда.

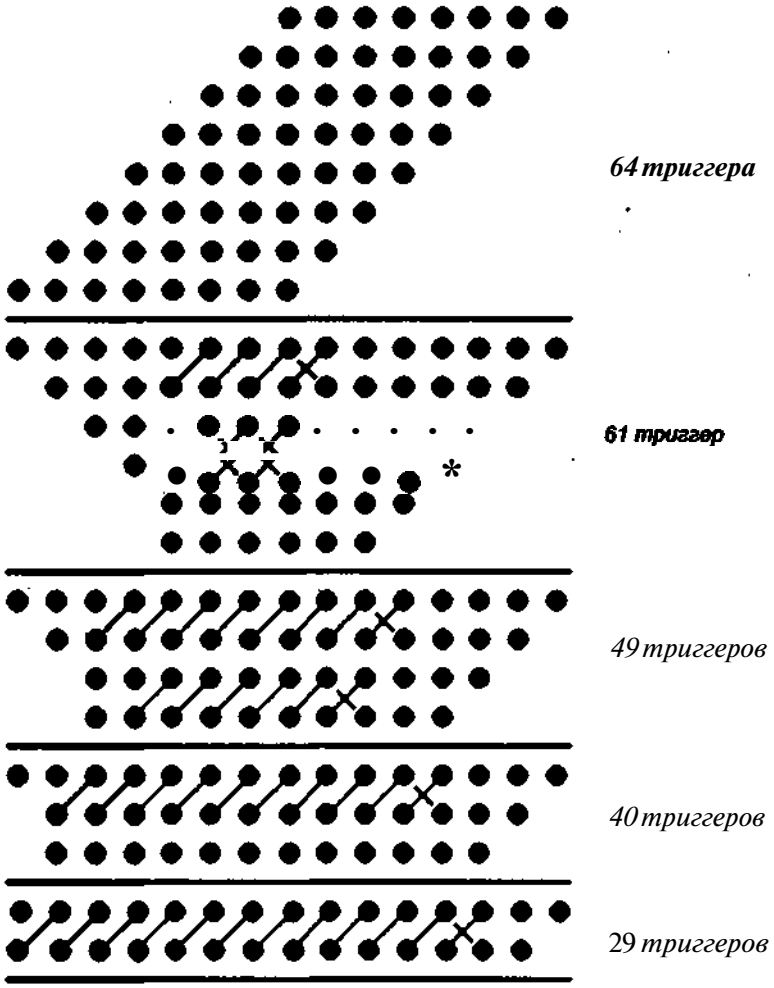


Рис. 7.45. Древовидный конвейеризированный умножитель со схемой Дадда

В правой части рисунка указано количество триггеров-защелок, необходимых в каждой ступени конвейера. Как видно, в умножителе Дадда 8x8 требуются 243 триггера, не считая дополнительных триггеров для конвейеризации последнего этапа сложения частичных произведений. Количество триггеров может быть сокращено за счет увеличения времени, выделяемого на выполнение операций в ступени конвейера. Это позволяет убрать некоторые из триггеров.

При конвейеризации умножителя на базе дерева Уоллеса требуется меньше триггеров-защелок, поскольку в этой схеме основное сжатие суммы частичных произведений происходит на более ранних этапах. Кроме того, для заключительного суммирования векторов сумм и переносов используется более «короткий» сумматор.

Рекурсивная декомпозиция операции умножения

Как правило, аппаратные умножители, построенные на рассмотренных принципах, имеют ограничение на число разрядов вводимых чисел. Умножитель повышенной разрядности можно получить из модулей меньшей разрядности, выстраивая так называемую *рекурсивную декомпозицию операции умножения*. Так, для построения умножителя 8x8 можно использовать четыре модуля типа 4x4. Множимое *A* разбивается на четыре старших *(A_h)* и четыре младших *(A)* разряда. Множитель *B* таким же образом разбивается на части B_h и *B1* Четыре модуля типа 4x4 вычисляют соответственно произведения $A_h x B_h$, $A_h x B_h$ $A_1 x B_h$, A_1 xB_1. На выходах модулей получаются восьмиразрядные результаты, которые соответствуют частичным произведениям в разрядах: 15-8,11-4, снова 11-4 и 7-0. Окончательный результат формируется путем суммирования этих четырех частичных произведений с учетом их положения в разрядной сетке (рис. 7.46).

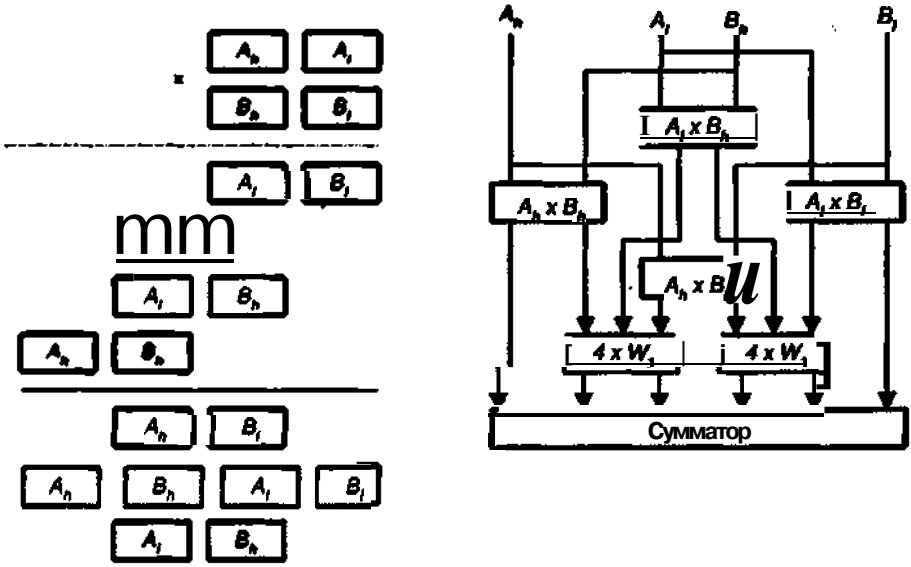


Рис. 7.49. Декомпозиция операции умножения

Целочисленное деление

Деление несколько более сложная операция, чем умножение, но базируется на тех же принципах. Основу составляет общепринятый способ деления с помощью операций вычитания или сложения и сдвига (рис. 7.47).

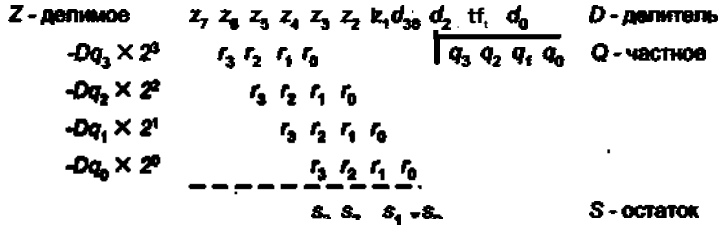


Рис. 7.47. Общая схема операции деления

Задача сводится к вычислению частного O и остатка S:

$$Q = \text{int}\left(\frac{Z}{D}\right), \qquad S = Z - QD, \quad S < D.$$

Деление выражается как последовательность вычитаний делителя сначала из делимого, а затем из образующихся в процессе деления частичных остатков (40). Делимое $Z(z_{2n-1}z_{2n-2}\ldots z_1z_0)$ обычно представляется двойным словом *(2n* разрядов), делитель $D(d_{n-1}d_{n-2}\ldots d_1d_0)$, частное $Q(q_{n-1}q_{n-2} — q_1q_0)$ и остаток $S(s_{n-1}s_{n-2}— s_1s_0)$ имеют разрядность и.

Операция выполняется за л итераций и может быть описана следующим образом:

$$S^{(i)} = 2S^{(i-1)} - q_{n-i}(2^nD), \text{ при } S^{(0)} = Z \text{ и } S^{(n)} = 2^nS;$$

$$q_{n-i} = \begin{cases} 1, \text{ если } (2S^{(i-1)} - 2^nD) \geq 0, \\ \text{O, если } (2S^{(i-1)} - 2^nD) < 0. \end{cases}$$

После n итераций получается

$$S^{(n)} = 2^nS^{(0)} - Q(2^nD) - 2^n[Z - (Q \times D)] - 2^nS.$$

Частное от деления 2n-разрядного числа на n-разрядное может содержать более, чем n разрядов. В этом случае возникает переполнение, из-за чего перед выполнением деления необходима проверка условия

$$Z < (2^n - 1)D + D = 2^nD.$$

Из выражения следует, что переполнения не будет, если число, содержащееся в старших n разрядах делимого, меньше делителя.

Помимо этого требования, перед началом операции необходимо исключить возможность ситуации деления на 0.

Реализовать деление можно двумя основными способами:

- с неподвижным делимым и сдвигаемым вправо делителем;

я с неподвижным делителем и сдвигаемым влево делимым.

Недостатком первого способа является потребность иметь в устройстве деления сумматор и регистр двойной длины. Второй способ позволяет строить дели-

тель с сумматором одинарной длины. Неподвижный делитель *D* хранится в регистре одинарной длины, а делимое *Z,* сдвигаемое относительно *D,* находится в двух таких же регистрах. Образующиеся цифры частного *Q* заносятся в освобождающиеся при сдвиге Z разряды одного из регистров Z

Ниже на. примере чисел без знака рассматриваются два основных алгоритма целочисленного деления.

Деление с восстановлением остатка

Наиболее очевидный алгоритм носит название алгоритма *деления с неподвижным делителем и восстановлением остатка.* В учебнике он представлен в силу того, что очень похож на общепринятый способ деления столбиком. Данный алгоритм может быть описан следующим образом:

1. Исходное значение частичного остатка полагается равным старшим разрядам делимого.
2. Частичный остаток удваивается путем сдвига на один разряд влево. При этом в освобождающийся при сдвиге младший разряд 40 заносится очередная цифра частного.
3. Из сдвинутого Ч0 вычитается делитель и анализируется знак результата вычитания.
4. Очередная цифра модуля частного равна единице, когда результат вычитания положителен, и нулю, если отрицателен. В последнем случае значение остатка восстанавливается до того значения, которое было до вычитания.
5. Пункты 2-4 последовательно выполняются для получения всех цифр модуля частного.

На рис. 7,48 показан процесс деления с восстановлением остатка, здесь число 41 делится на 8.

Деление без восстановления остатка

Недостаток затронутого алгоритма заключается в необходимости выполнения на отдельных шагах дополнительных операций сложения для восстановления частичного остатка. Это увеличивает время выполнения деления, которое в этом случае может меняться в зависимости от конкретного сочетания кодов операндов. В силу указанных причин реальные делители строятся на основе алгоритма *деления с неподвижным делителем без восстановления остатка.* Приведем описание этого алгоритма.

1, Исходное значение частичного остатка полагается равным старшим разрядам делимого.
2, Частичный остаток удваивается путем сдвига на один разряд влево. При этом в освобождающийся при сдвиге младший разряд Ч0 заносится очередная цифра частного.
3, Из сдвинутого частичного остатка вычитается делитель, если остаток положителен, и к сдвинутому частичному остатку прибавляется делитель, если остаток отрицательный.

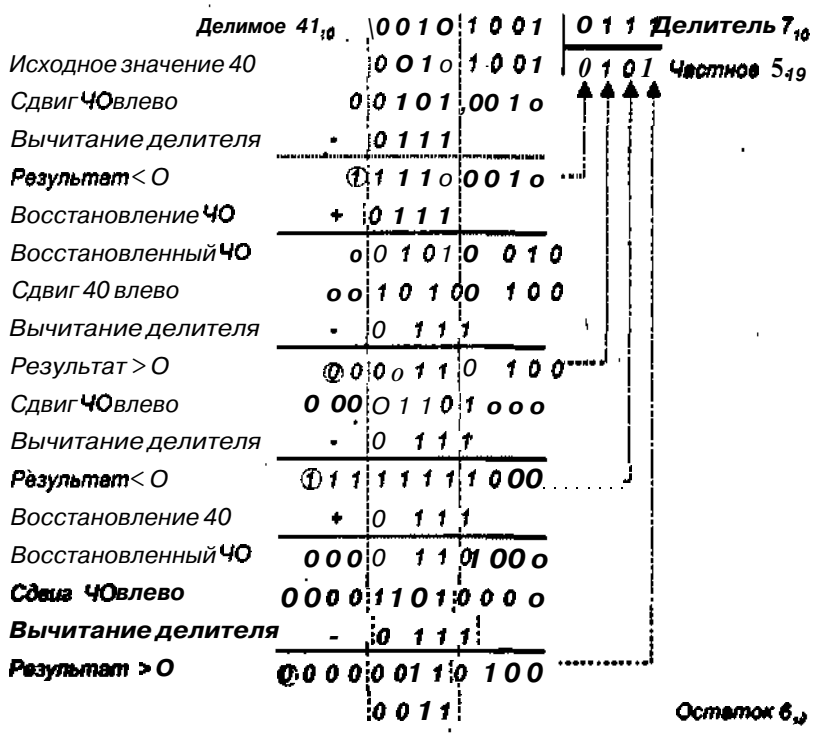


Рис. 7,48. Пример деления с восстановлением остатка

4. Очередная цифра модуля частного равна единице, когда результат вычитания положителен, и нулю, если он отрицателен.
5. Пункты 2-4 последовательно выполняются для получения всех цифр модуля частного.

Как видим, пункты 1,2,5 полностью совпадают с соответствующими пунктами предыдущего алгоритма деления.

Процесс деления без восстановления остатка для ранее рассмотренного примера демонстрируется на рис. 7.49.

Деление чисел со знаком

Как и в случае умножения, деление чисел со знаком может быть выполнено путем перехода к абсолютным значениям делимого и делителя, с последующим присвоением частному знака «плюс» при совпадающих знаках делимого и делителя либо «минус» — в противном случае.

Деление чисел, представленных в дополнительном коде, можно осуществлять не переходя к модулям. Рассмотрим необходимые для этого изменения в алгоритме без восстановления остатка.

Так как делимое и делитель не обязательно имеют одинаковые знаки, то действия с частичным остатком (прибавление или вычитание *D)* зависят от знаков остатка и делителя и определяются содержимым табл. 7,5:

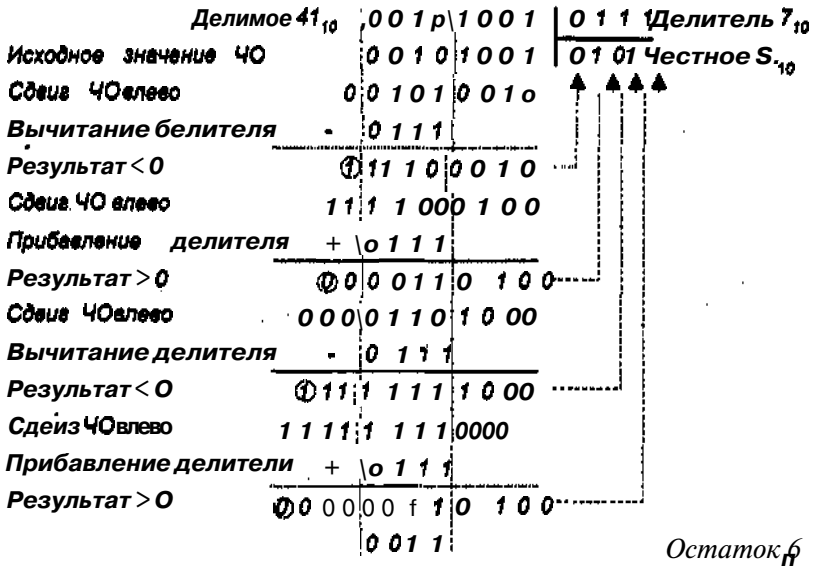


Рис. 7.49. Пример деления без восстановления остатка

Знак остатка	Знак делителя	Действие
+	+	Вычитание делителя
+	-	Прибавление делителя
-	+	Прибавление делителя
-	-	Вычитание делителя

И Если знак остатка совпадает со знаком делителя, то очередная цифра частного — 1, иначе — 0.

- *Если Z>0* и Ж 0, частное необходимо увеличить на 1.
- Если Z < 0 и *D* > 0, то при ненулевом остатке от деления частное нужно увеличить на единицу.
- Если Z < 0 и *D* < 0, то при нулевом остатке от деления частное нужно увеличить на единицу.

Остаток всегда приводится к положительному числу, то есть если по завершении деления он отрицателен, к нему следует прибавить модуль делителя.

Устройство деления

Рассмотренный алгоритм деления без восстановления остатка может быть реализован с помощью устройства, схема которого приведена на рис. 7.50.

Процедура начинается с занесения делимого в 2n-разрядный регистр делимого (РДМ) и делителя в n-разрядный регистр делителя (РДТ). В счетчик цикла (СЧЦ - на схеме не показан), служащий для подсчета количества полученных цифр частного, помещается исходное значение, равное *n*.

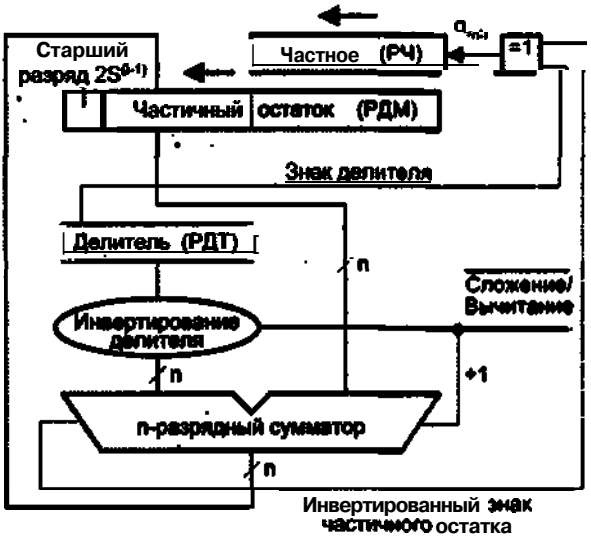


Рис. 7.50. Схема деления по алгоритму без восстановления остатка

На каждом шаге содержимое регистра делимого (РДМ) и регистра частного (РЧ) сдвигается на один разряд влево. В зависимости от сочетания знаков частичного остатка и делителя определяется значение очередной цифры частного и требуемое действие: вычитание или прибавление делителя. Вычитание делителя производится посредством прибавления дополнительного кода делителя. Преобразование в дополнительный код осуществляется за счет передачи делителя на вход сумматора обратным (инверсным) кодом с последующим добавлением единицы к младшему разряду сумматора.

Описанная процедура повторяется до исчерпания всех цифр делимого, о чем свидетельствует нулевое содержимое счетчика циклов (содержимое СЧЦ уменьшается на единицу после каждой итерации). По окончании операции деления частное располагается в регистре частного, а в регистре делимого будет остаток от деления.

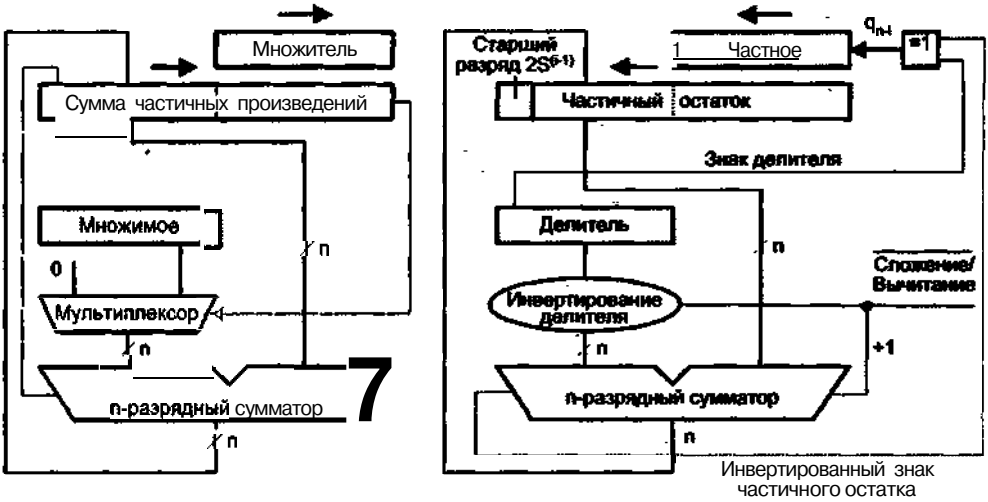
На заключительном этапе, если это необходимо, производится корректировка полученного результата, как это предусматривает алгоритм деления чисел со знаком.

На практике для накопления и хранения частного вместо отдельного регистра используют освобождающиеся в процессе сдвигов младшие разряды регистра делимого.

Комбинированное устройство умножения/деления

Сходство процедур умножения и деления находит свое отражение в близости структур соответствующих устройств (рис. 7.51).

Из подобия процедур вытекает очевидная идея реализации обеих операций с помощью единых технических средств, в виде комбинированного устройства умножения-деления (рис. 7.52).



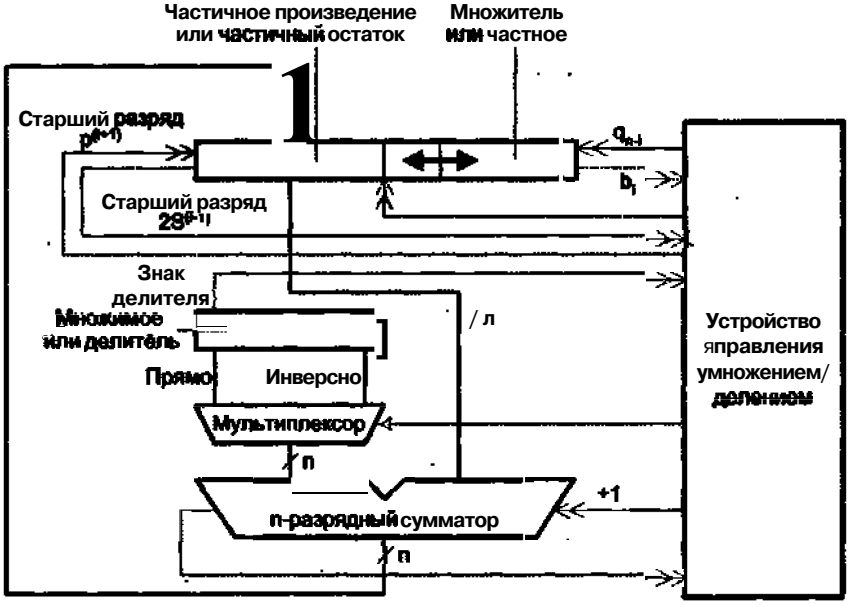


Рис. 7.52. Комбинированное устройство умножения/деления

Видим, что для хранения операндов и результатов используются общие регистры. Усложнение связано, главным образом, с устройством управления, функции которого существенно расширяются, что, естественно, требует определенного увеличения аппаратурных затрат.

Ускорение целочисленного деления

Следует отметить, что операция деления предоставляет не слишком много путей для своей оптимизации по времени. Тем не менее определенные возможности для убыстрения деления существуют-, и их можно свести к следующим:

- замена делителя обратной величиной, с последующим ее умножением на делимое;
- сокращение времени вычисления частичных остатков в традиционных методах деления (с восстановлением или без восстановления остатка) за счет ускорения операций суммирования (вычитания);
- сокращение времени вычисления за счет уменьшения количества операций суммирования (вычитания) при расчете Ч0;
- вычисление частного в избыточной системе счисления.

За исключением первого из перечисленных подходов все прочие фактически являются модификациями традиционного способа деления.

Замена деления умножением на обратную величину

В предыдущем разделе было показано, что операцию умножения можно производить сравнительно быстро, если взять на вооружение комбинационные схемы параллельного умножения. Данное обстоятельство можно использовать, заменив операцию деления на *D* умножением на

$$\frac{1}{D} \cdot Q = \frac{Z}{D} = Z_X \frac{1}{D}.$$

В этом случае проблема сводится к эффективному вычислению *1/D.* Обычно задача решается одним из двух методов: с помощью ряда Тейлора или метода Ньютона-Рафсона. В обоих случаях основное время расходуется на умножение, поэтому рассматриваемый метод ускорения деления имеет смысл при наличии быстрых схем умножения.

При реализации первого метода делитель *D* представляется в виде: $D - 1 + X$. Тогда для двоичного представления *D* можно записать:

$$\frac{1}{D} = (1 - X) \times (1 + X^2) \times (1 + X^4) \times (1 + X^8) \times (1 + X^{16}) \ldots$$

Метод был использован в модели 91 вычислительной машины IBM 360 для вычисления 32-разрядной величины *1/D.* Возможные значения сомножителей в правой части выражения извлекались из таблицы емкостью 28 байт, хранящейся в памяти. Операция вычисления *1/D* требует шести умножений.

Вычисление величины *1/D* методом Ньютона-Рафсона сводится к нахождению корня уравнения

$$f(X) = \frac{1}{X} - D = 0,$$

то есть *X- 1/D.* Решение может быть получено с привлечением рекуррентного соотношения: $X_{i+1} = X_i(2 - X_i D)$ •. Количество итераций определяется требуемой точностью вычисления *1/D.* Реализация метода для n-разрядных чисел требует $2\,\mathrm{int}(\log_2 n) - 1$ операций умножения.

В общем, замена операции деления на умножение более характерна для чисел с плавающей запятой.

Ускорение вычисления частичных остатков

Возможности данного подхода весьма ограничены и связаны в основном с ускорением операций сложения (вычитания). Способы достижения этой цели ничем не отличаются от тех, что применяются, например, при выполнении умножения. Это различные приемы для убыстрения распространения переноса, матричные схемы сложения и т. п. В частности, для вычисления частичных остатков может быть применена матричная схема, показанная на рис. 7.53.

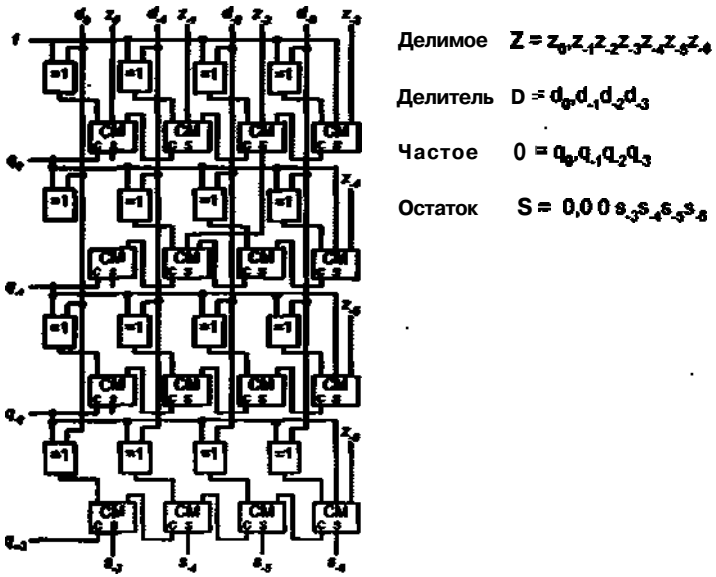


Рис. 7.53. Матричное устройство деления для алгоритма без восстановления остатка

Представленная схема реализует алгоритм деления без восстановления остатка для правильных дробей, что типично в операциях над мантиссами чисел в форме с плавающей запятой. Нетрудно заметить, что схеме присущ длинный тракт распространения переноса, что явно не способствует сокращению времени деления. Таким образом, матричное исполнение деления нельзя считать очень эффективным в плане быстродействия, хотя данный метод и выигрывает в сравнении с традиционным устройством деления. Основное достоинство матричной схемы заключается в ее регулярности, что удобно при реализации устройства в виде интегральной микросхемы.

Алгоритм S R T

В основе третьей группы методов ускорения операции деления, согласно приведенной выше классификации, лежит так называемый алгоритм SRT [77,184,214]. Свое название алгоритм получил по фамилиям авторов (Sweeney, Robertson, Tocher), разработавших его независимо друг от друга приблизительно в одно и то же

время. Этот алгоритм представляет собой модификацию деления без восстановления остатка. В стандартной процедуре на каждом шаге помимо сдвига частичного остатка производится прибавление либо вычитание делителя. В SRT-алгоритме сдвиг Ч0 также имеется в каждой итерации, однако сложение или вычитание, в зависимости от получающегося Ч0, на отдельных шагах может не выполняться, что, естественно, позитивно влияет на быстродействие деления.

Алгоритм был ориентирован на операции над мантиссами чисел с плавающей запятой и опирается на то обстоятельство, что мантиссы в таких числах нормализованы. Впервые SRT-алгоритм был реализован в модели 91 вычислительной машины IBM 360. В настоящее время он широко применяется в блоках обработки чисел с плавающей запятой, в частности в микропроцессорах фирмы Intel.

Сначала рассмотрим алгоритм применительно к положительным целым числам. Делимое представляется (2n + 1)-разрядным числом, а делитель — «-разрядным. Процедура деления начинается с удаления в делителе всех нулей, предшествующих старшей единице, то есть с операции, аналогичной нормализации мантиссы в числах с плавающей запятой. По той же аналогии будем в дальнейшем условно называть эту операцию нормализацией. Исключение k предшествующих нулей реализуется за счет сдвига делителя влево на $к$ разрядов. На аналогичное число разрядов влево сдвигается и делимое. Далее выполняются n итераций, в которых вычисляются цифры частного и частичные остатки. Действия, выполняемые на i-й итерации, можно описать следующим образом:

$$q_i = \begin{cases} 1, \text{ если } 2S^{(i-1)} \geq D; \\ 0, \text{ если } -D \leq 2S^{(i-1)} < D; \\ -1, \text{ если } 2S^{(i-1)} < -D, \end{cases}$$

$$S^{(i)} = 2S^{(i-1)} - q_i D.$$

Обратим внимание на то, что частное представляется в системе счисления, отличной от двоичной. Это означает, что цифры частного могут иметь больше, чем два значения 0 и 1. В рассматриваемом случае — 1, 0, 1.

По завершении всех n итераций, если последний остаток отрицателен, выполняется коррекция этого остатка и полученного частного, для чего к остатку прибавляется делитель, а из частного вычитается единица с весом младшего разряда.

Последний момент в алгоритме - преобразование частного из системы {-1,0,1} в систему {0,1}, то есть в обычную двоичную систему.

На практике это выливается в следующую процедуру (при объяснении будем ссылаться на схему деления без восстановления остатка, приведенную на рис. 7.50). Делимое и делитель, представленные в дополнительном коде, размещаются в регистре делимого (РДМ) и делителя (РДТ) соответственно. Дальнейшие действия можно описать следующим образом.

1. Если в делителе D имеются k предшествующих нулей (при $D > 0$) или предшежимого РДМ и РДТ влево на k разрядов.

2. Для i от 0 до n - 1:

- если три старших цифры частичного остатка в РДМ совпадают, то $q_i = 0$ и производится сдвиг содержимого РДМ на один разряд влево;
- если три старших цифры частичного остатка в РДМ не совпадают, а сам Ч0 отрицателен, то q_i - 1, делается сдвиг содержимого РДМ на один разряд влево и к Ч0 прибавляется делитель;
- если три старших цифры частичного остатка в РДМ не совпадают, а сам Ч0 положителен, то q_i - 1, выполняется сдвиг содержимого РДМ на разряд влево и из Ч0 вычитается делитель.

3. Если после завершения пункта 2 остаток отрицателен, то производится коррекция (к остатку прибавляется делитель, а из частного вычитается единица).
4. Остаток сдвигается вправо на *к* разрядов.

Описанную процедуру иллюстрирует пример, приведенный на рис. 7.54.

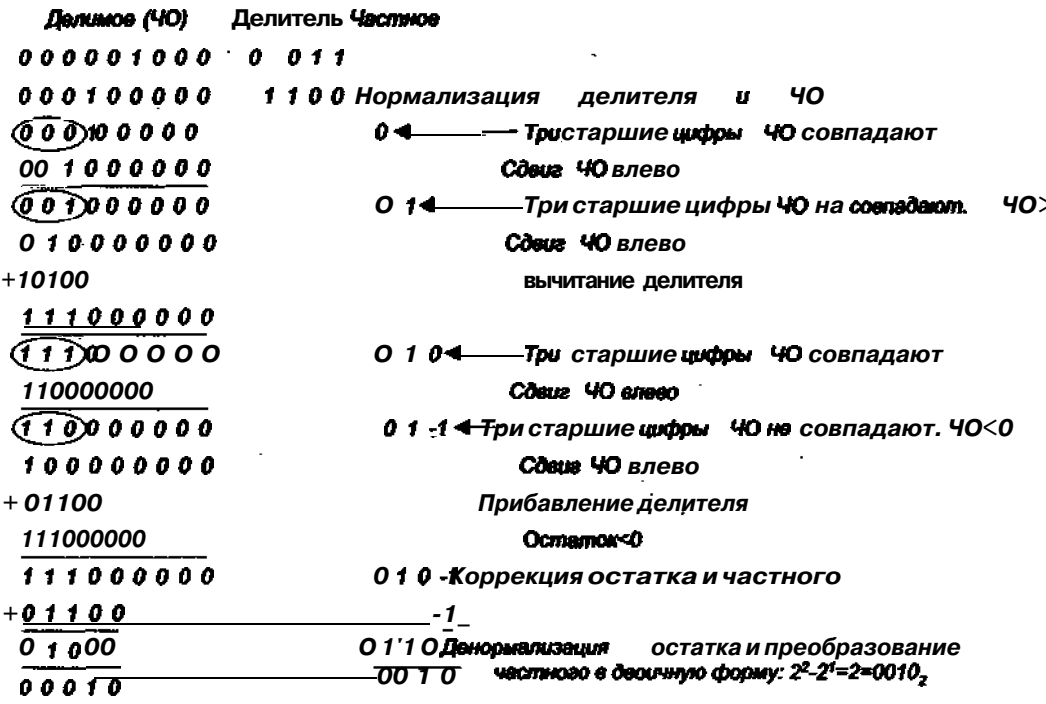


Рис. 7.54. Пример деления целых чисел по алгоритму S R T

На первом шаге для удаления предшествующих нулей делитель сдвигается на два разряда влево. Аналогично поступают и с 4 0, который вначале совпадает с делимым. Далее выполняется процедура, описанная выше в пункте 2. Операция вычитания *D* обеспечивается прибавлением делителя с противоположным знаком. Поскольку по завершении операции остаток отрицателен, производится его коррекция путем прибавления *D*. Одновременно частное уменьшается на единицу (эта операция показана в системе {-1,0,1}, в которой представлено частное). Наконец,

на последнем шаге форма представления частного меняется, переходят к представлению в стандартной двоичной системе.

В стандартном алгоритме деления без восстановления остатка помимо сдвига в каждой итерации выполняется операция сложения или вычитания. В варианте SRT, в зависимости от кодов операндов в отдельных итерациях, достаточно только сдвига, что, безусловно, ускоряет процесс деления. Согласно статистическим данным, в среднем число сложений и вычитаний при использовании этого алгоритма сокращается в 2,67 раза.

Деление в избыточных системах счисления

Наиболее распространенные методы ускорения операции деления основаны на применении алгоритмов, где частное представляется в системе счисления, отличной от двоичной. Это означает, что цифры частного могут иметь больше, чем два значения, например {-1,0,1}, как это было в алгоритме умножения Бута, или {-2, -1,0,1,2}. В таких системах одно и то же число может быть записано несколькими способами, из-за чего системы называют избыточными. Очередная цифра частного в избыточной системе счисления, в зависимости от базы этой системы, соответствует двум или более цифрам в двоичном представлении частного, и для нужного количества двоичных цифр частного и остатка требуется меньше итераций. В то же время реализация такого подхода ведет к усложнению аппаратуры делителя, в частности надстраивается логика определения операции, выполняемой в очередной итерации. Для этой цели в состав устройства деления включается специальная память, хранящая таблицу, определяющую необходимые действия, в зависимости от текущей комбинации цифр в частичном остатке и делителе. Тем не менее выигрыш в быстродействии оказывается решающим моментом. Так, в микропроцессорах Pentium при делении мантисс чисел с плавающей запятой используется алгоритм SRT с базой 4, то есть частное сначала вычисляется с использованием цифр -2, -1, 0, 1, 2 с последующим преобразованием результата к стандартному двоичному представлению. В этом варианте выбор очередной цифры частного производится с помощью таблицы, состоящей из отдельных секций. Конкретную секцию определяют четыре старшие цифры делителя (после его нормализации). Входом в секцию служат шесть старших цифр частичного остатка. Ч0 в каждой итерации сдвигается не на один, а на два разряда, то есть число итераций сокращается вдвое. Известны варианты делителей, где берется еще большее основание системы счисления, в частности 8 и 16. В этом случае логика работы устройства существенно усложняется.

Операционные устройства с плавающей запятой

Операции над числами в формате с плавающей запятой (ПЗ) имеют существенные отличия от аналогичных операций целочисленной арифметики, поэтому их обычно реализуют с помощью самостоятельного операционного устройства. Как и целочисленное ОПУ, операционное устройство для чисел в формате ПЗ как ми-

нимум должно обеспечивать выполнение четырех арифметических действий: сложения, вычитания, умножения и деления.

После принятия стандарта IEEE 754 негласным требованием ко всем ВМ является обеспечение операций с числами, представленными в форматах, которые определены данным стандартом. Это требование не исключает использования в конкретной ВМ также иных форматов чисел с ПЗ, что обычно связано с сохранением совместимости с предыдущими моделями данной вычислительной машины.

Напомним основные положения записи чисел в стандарте IEEE 754. Мантиссы чисел M представляются в нормализованном виде, при этом действует прием скрытого разряда, когда старшая цифра мантиссы, всегда равная единице, в записи числа отсутствует, то есть в поле мантиссы старшей является вторая старшая цифра нормализованной мантиссы.

В отличие от общепринятого условия нормализации $S - /M/ < 1$, в стандарте IEEE 754 используется условие $1 - /M/ < 2$.

Запись числа содержит смешенный порядок, то есть порядок, увеличенный на величину смешения, которое в стандарте IEEE 754 для одинарного формата равно 127, а для двойного - 1023.

С учетом перечисленных особенностей арифметическую операцию над числами в формате с плавающей запятой можно записать в виде:

$$\pm Z_M \times 2^{Z_{СП}} = (\pm X_M \times 2^{X_{СП}}) \; 0 \; (\pm Y_M \times 2^{Y_{СП}}),$$

где X_M, Y_M, Z_M - нормализованные мантиссы операндов и результата; $X_{СП}$, $Y_{СП}$, $Z_{СП}$ — смещенные порядки операндов и результата; 0 — знак арифметической операции.

При всех различиях в выполнении разных арифметических операций подготовительный и заключительный этапы во всех случаях совпадают, в силу чего их имеет смысл рассмотреть отдельно.

Подготовительный этап

Первой особенностью операционных устройств для чисел с плавающей запятой является то, что в них операции над тремя составляющими чисел с ПЗ (знаками, мантиссами и порядками операндов) выполняются раздельно: блоком обработки знаков (БОЗ), блоком обработки порядков (БОП) и блоком обработки мантисс (БОМ). Для хранения операндов и результата в ОПУ предусмотрены соответствующие регистры. Хотя эти регистры могут быть физически реализованы в виде единых устройств, каждый из них логично рассматривать как совокупность трех регистров: знака, порядка и мантиссы. Таким образом, на этапе загрузки операндов в регистры ОПУ осуществляется «распаковка» чисел с ПЗ, их разбиение на три составляющие. В ходе такой распаковки в старшем разряде регистра мантиссы восстанавливается единица, которая в записи числа отсутствовала (была скрыта).

На предварительном этапе может быть также выполнена проверка на равенство нулю одного или обоих операндов (в стандарте IEEE 754 для представления нулевого значения используется такая запись числа, в которой нулю равны все разряды порядка). Это позволяет исключить ненужные операции. Так, в операциях умножения и деления, если нулю равны множитель, множимое или делимое, в качестве результата сразу же можно принять нулевое значение, обойдя предписанные данными операциями действия.

Заключительный этап

Действия на завершающем этапе выполнения любой арифметической операции идентичны и сводятся к выявлению нулевого значения мантиссы (потери значимости мантиссы), нормализации мантиссы, выявлению отрицательного переполнения порядка, "упаковке" составляющих результата.

Нулевое значение мантиссы может получиться в результате операции, например при сложении или вычитании мантисс. Второй причиной может стать сдвиг мантиссы вправо для устранения переполнения. В обоих случаях имеет место ситуация *потери значимости мантиссы,* и результат операции принимается равным нулю. Для стандарта IEEE 754 это означает, что все цифры порядка результата необходимо обнулить, а также то, что нормализацию мантиссы результата производить не нужно.

Нормализация мантиссы результата сводится к последовательному ее сдвигу влево до тех пор, пока старшую позицию не займет единица. Каждый сдвиг сопровождается уменьшением на единицу порядка результата. В ходе уменьшения порядок может стать отрицательным, что для смещенных порядков свидетельствует о получение числа, непредставимого в данном формате. В такой ситуации результат принимается равным нулю и одновременно формируется признак *потери значимости порядка*.

В завершение мантисса результата округляется и, если это предусмотрено форматом ПЗ, из нее удаляется скрытый разряд.

В последней фазе осуществляется "упаковка" всех составляющих результата (знака, порядка и мантиссы), после чего сформированный результат заносится в выходной регистр ОПУ.

Сложение и вычитание

В арифметике с плавающей запятой сложение и вычитание - более сложные операции, чем умножение и деление. Обусловлено это необходимостью выравнивания порядков операндов. Алгоритм сложения и вычитания включает в себя следующие основные фазы:

1. Подготовительный этап.
2. Определение операнда, имеющего меньший порядок, и сдвиг его мантиссы вправо на число разрядов, равное разности порядков операндов.
3. Приравнивание порядка результата большему из порядков операндов.
4. Сложение или вычитание мантисс и определение знака результата.
5. Проверку на переполнение.
6. Заключительный этап.

Операции предшествует вышеописанный подготовительный этап, в ходе которого операнды «распаковываются» и помещаются в регистры ОПУ,

Сложение и вычитание выполняются идентично, но в случае вычитания необходимо изменить знак второго операнда на противоположный. Далее производится проверка с целью выяснения, не равен ли нулю один из операндов. Если это имеет место, в качестве результата сразу берется другой операнд.

В следующей фазе осуществляется такое преобразование одного из исходных чисел, чтобы порядки обоих операндов стали равны. Для пояснения рассмотрим пример сложения десятичных чисел с ПЗ:

$$123 \times 100 + 456 \times 10^{-2}.$$

Очевидно, что непосредственное сложение мантисс недопустимо, поскольку цифры мантисс, имеющие одинаковый вес, должны располагаться в эквивалентных позициях. Так, цифра 4 во втором числе должна суммироваться с цифрой 3 в первом. Этого можно добиться, если записать второе число так, чтобы порядки обоих чисел были равны:

$$123 \times 10^{0} + 456 \times 10^{-2} = 123 \times 10^{0} + 4{,}56 \times 10^{0} - 127{,}56 \times 10^{0}.$$

Выравнивания порядков можно достичь сдвигом мантиссы меньшего из чисел вправо, с одновременным увеличением порядка этого числа, либо сдвигом мантиссы большего из чисел влево и уменьшением его порядка. Оба варианта сопряжены с потерей цифр мантиссы, но выгоднее сдвигать меньшее из чисел, так как при этом теряются младшие разряды мантиссы. Таким образом, выравнивание порядков операндов реализуется путем сдвига мантиссы меньшего из чисел на один разряд вправо с одновременным увеличением порядка этого числа на единицу. Действия повторяются до совпадения порядков. Если в процессе сдвига мантисса обращается в 0, в качестве результата операции берется другой операнд.

Следующая фаза — сложение мантисс с учетом их знаков, что при одинаковых знаках мантисс может привести к переполнению. В последнем случае мантисса результата сдвигается вправо на один разряд, а порядок результата увеличивается на единицу. Это, в свою очередь, чревато переполнением поля порядка. Тогда операция прекращается и формируется признак переполнения, сопровождаемый соответствующим предупреждением (обычно в виде сигнала прерывания).

Далее выполняется описанный выше заключительный этап.

• В отличие от целочисленной арифметики, в операциях с ПЗ сложение и вычитание производятся приближенно, так как при выравнивании порядков происходит потеря младших разрядов одного из слагаемых. В этом случае погрешность всегда отрицательна и может доходить до единицы младшего разряда.

Умножение

На начальном этапе умножения чисел с ПЗ производится проверка на равенство нулю одного из сомножителей. Если один из операндов равен нулю, в качестве результата выдается 0, представленный в данном формате чисел с ПЗ. Следующий шаг — сложение порядков. Если в рассматриваемом формате используется смещенный порядок, то в полученной сумме будет содержаться удвоенное смещение, поэтому из нее необходимо вычесть величину смещения. Результатом действий с порядками может стать как переполнение порядка, так и потеря значимости. В обоих случаях выполнение операции прекращается и выдается соответствующее сообщение (возникает прерывание).

Если порядок результата лежит в допустимых границах, на следующем шаге производится перемножение мантисс с учетом их знака, которое выполняется так же, как для чисел с фиксированной запятой. При размещении произведения мантисс в разрядной сетке необходимо учитывать, что мантиссы представлены не це-

лыми числами, а правильными дробями. Хотя результат умножения мантисс имеет удвоенную по сравнению с операндами длину, он округляется до длины поля мантиссы.

На последнем Шаге производится нормализация и компоновка результата, аналогично тому, как это имеет место при сложении и вычитании. Отметим, что при нормализации результата возможно переполнение порядка.

Деление

Сначала также проводится проверка на 0. Если нулю равен делитель, в зависимости от реализации выдается сообщение о делении на 0, либо в качестве результата принимается бесконечность (для этого в некоторых форматах ПЗ имеется специальная кодовая комбинация). Когда нулю равно делимое, результат также принимается равным нулю.

Далее выполняется вычитание порядка делителя из порядка делимого, что приводит к удалению смещения из порядка результата. Следовательно, для получения смещенного порядка результата к разности должно быть добавлено смещение. После выполнения этих действий необходима проверка на переполнение порядков и потерю значимости.

Следующий шаг — деление мантисс, за которым идут нормализация, округление и компоновка числа из мантиссы и порядка.

Реализация логических операций

Помимо арифметических действий ОПУ любой вычислительной машины предполагает выполнение основных логических операций и сдвигов. Чаще всего такие операции реализуются дополнительными схемами, входящими в состав целочисленных ОПУ.

К базовым логическим операциям относятся: логическое отрицание (НЕ), логическое сложение (ИЛИ) и логическое умножение (И). Этот набор, как правило, дополняют операцией сложения по модулю 2 (исключающее ИЛИ).

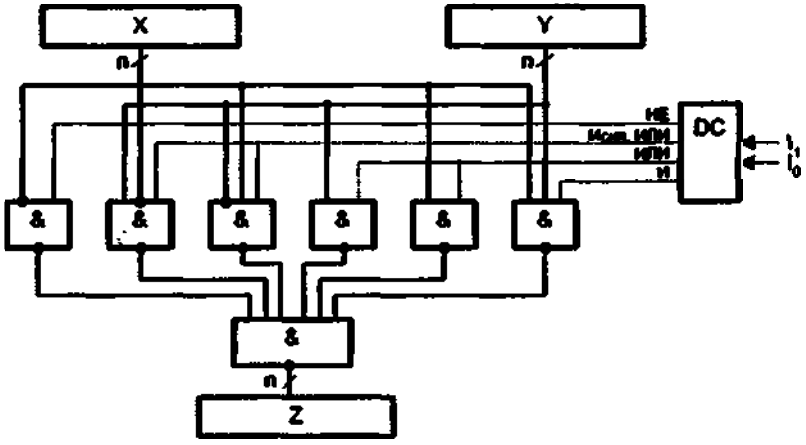


Рис. 7.55. Структура операционного блока для выполнения логических операций

Булева переменная в ВМ представляется одним битом, однако на практике логические операции в ОПУ выполняются .сразу над совокупностью логических переменных, объединенных в рамках одного байта или машинного слова, причем над всеми битами этого слова выполняется одна и та же операция. Поскольку каждый бит совокупности логических переменных рассматривается как независимая переменная, никакие переносы между разрядами не формируются. Операционный блок (ОПБ) для выполнения логических операций обеспечивает реализацию всех перечисленных логических операций. Возможная структура подобного ОПБ показана на рис. 7.55.

Выбор нужной операции осуществляется с помощью двухразрядного управляющего кода L (//). С учетом управляющего кода выходная функция Z может быть описана выражением:

$$Z = \overline{l}_1 \wedge \overline{l}_0 \wedge \overline{X} \vee \overline{l}_1 \wedge l_0 \wedge (\overline{X} \wedge Y \vee X \wedge \overline{Y} l_1 \vee \wedge \overline{l}_0 \wedge (X \vee Y) \vee l_1 \wedge l_0 \wedge (X \wedge Y).$$

Контрольные вопросы

1. Охарактеризуйте состав операционных устройств, входящих в АЛУ. Из каких соображений и каким образом он может изменяться?
2. Поясните понятие "операционные устройства с жесткой структурой"-. В чем заключается жесткость их структуры? Каковы их достоинства и недостатки?
3. Чем обусловлено название операционных устройств с магистральной структурой? Сравните магистральные структуры с жесткими структурами, выделяя достоинства, недостатки и область применения.
4. Дайте развернутую характеристику классификации операционных устройств с магистральной структурой. Поясните достоинства и недостатки «минимального» и «максимального» вариантов.
5. Поясните функциональный состав параллельного операционного блока магистрального ОПУ. Каким образом можно минимизировать количество внешних связей этого блока? Ответ сопроводите конкретным примером.
6. Чем обусловлена Специфика целочисленного сложения и вычитания? Какую роль играет в них дополнительный код? К чему бы привел отказ от дополнительного кода? Ответ поясните на примерах. Как выявляется переполнение в этих операциях?
7. Сформулируйте достоинства, недостатки и область применения четырех вариантов целочисленного «традиционного» умножения. Как учитываются знаки сомножителей?
8. Охарактеризуйте суть двух групп логических методов ускорения умножения.
9. Попарно сравните алгоритм Бута, модифицированный алгоритм Бута, алгоритм Лемана.
10. Разработайте алгоритм умножения с обработкой за шаг трех разрядов множителя.
11. Поясните суть аппаратных методов ускорения умножения, выделив три возможных подхода.

12. Сравните умножители Брауна, Бо-Вули, Пезариса. Чем они схожи и в чем отличаются друг от друга?
13. В чем заключается основная идея древовидных умножителей? Каковы особенности их организации?
14. Сформулируйте, в чем состоит сходство и различие дерева Уоллеса, дерева Дадда и перевернутого ступенчатого дерева.
15. С какой целью и каким образом выполняется конвейеризация матричных и древовидных умножителей? Приведите (и поясните) конкретные примеры.
16. На конкретном примере объясните, как можно нарастить разрядность аппаратного умножителя.
17. Сравните организацию целочисленного деления с восстановлением остатка и без восстановления остатка. Как учитываются при делении знаки операндов?
18. Обоснуйте возможность совмещения структур умножителя и делителя. Опишите объединенную структуру.
19. Сформулируйте четыре пути ускорения целочисленного деления. Сравните между собой их возможную реализацию.
20. Какие из операций с плавающей запятой считаются наиболее сложными? Ответ обоснуйте на конкретных примерах.
21. В чем состоит специфика умножения с плавающей запятой? Поясните эту специфику на примере.
22. Разработайте серию примеров для иллюстрации всех особенностей деления с плавающей запятой.
23. Создайте структуру операционного блока для выполнения как сложения/вычитания, так и базового набора логических операций. Обоснуйте каждый элемент этой структуры.

Глава 8
Системы ввода/вывода

Помимо центрального процессора (ЦП) и памяти, третьим ключевым элементом архитектуры ВМ является *система ввода/вывода* (СВВ). Система ввода/вывода призвана обеспечить обмен информацией между ядром ВМ и разнообразными внешними устройствами (ВУ). Технические и программные средства СВВ несут ответственность за физическое и логическое сопряжение *ядра вычислительной машины и ВУ.*

В процессе эволюции вычислительных машин системам ввода/вывода по сравнению с прочими элементами архитектуры уделялось несколько меньшее внимание. Косвенным, подтверждением этого можно считать, например, то, что многие программы контроля производительности (бенчмарки) вообще не учитывают влияние операций ввода/вывода (В/ВЫВ) на эффективность ВМ. Следствием подобного отношения стал существенный разрыв в производительности процессора и памяти, с одной стороны, и скоростью ввода/вывода - с другой.

Технически система ввода/вывода в рамках ВМ реализуется комплексом *модулей ввода/вывода* (МВБ). Модуль ввода/вывода выполняет сопряжение ВУ с ядром ВМ и различные коммуникационные операции между ними. Две основные функции МВБ:

- обеспечение интерфейса с ЦП и памятью *("большой" интерфейсу,*
- обеспечение интерфейса с одним или несколькими периферийными устройствами *("малый" интерфейс).*

Анализируя архитектуру известных ВМ, можно выделить три основных способа подключения СВВ к ядру процессора (рис. 8.1).

В варианте с раздельными шинами памяти и ввода/вывода (см. рис. *8.1, а)* обмен информацией между ЦП и памятью физически отделен от ввода/вывода, поскольку обеспечивается полностью независимыми шинами. Это дает возможность осуществлять обращение к памяти одновременно с выполнением ввода/вывода. Кроме того, данный архитектурный вариант ВМ позволяет специализировать каждую из шин, учесть формат пересылаемых данных, особенности синхронизации обмена и т. п. В частности, шина ввода/вывода, с учетом характеристик реальных ВУ, может иметь меньшую пропускную способность, что позволяет снизить затраты на ее реализацию. Недостатком решения можно считать большое количество точек подключения к ЦП.

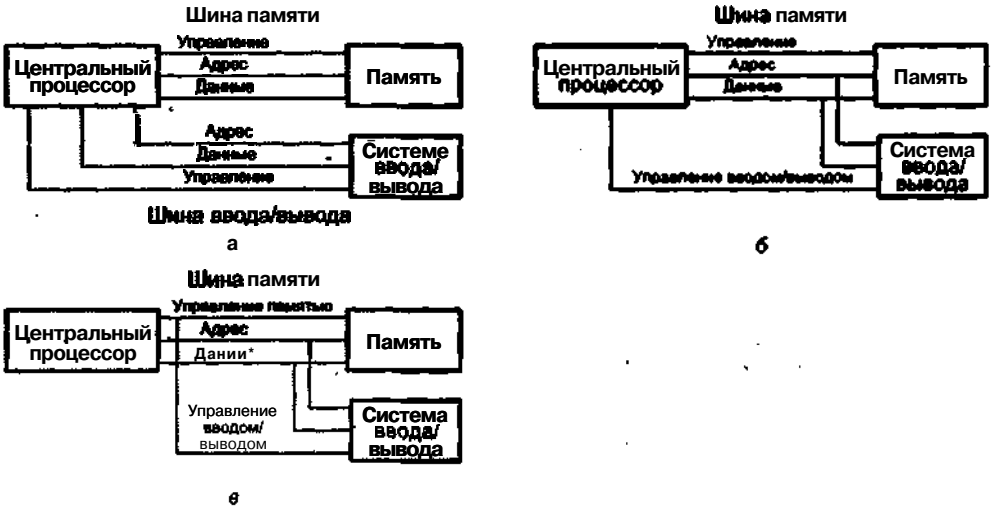


Рис. 8.1 . Место системы ввода/вывода в архитектуре вычислительной машины: *а* — раздельными шинами памяти и ввода/вывода; б —с совместно используемыми линиями данных и адреса; *в* — подключение на общих правах с процессором и памятью

Второй вариант — с совместно используемыми линиями данных и адреса (см. рис. 8.1, *б).* Память и СВВ имеют общие для них линии адреса и линии данных, разделяя их во времени. В то же время управление памятью и СВВ, а также синхронизация их взаимодействия с процессором осуществляются независимо по раздельным линиям управления. Это позволяет учесть особенности процедур обращения к памяти и к модулям ввода/вывода и добиться наибольшей эффективности доступа к ячейкам памяти и внешним устройствам.

Последний тип архитектуры ВМ предполагает подключение СВВ к системной шине на общих правах с процессором и памятью (см. рис. 8,1, *в).* Преимущества и недостатки такого подхода обсуждались при рассмотрении вопросов организации шин (глава 4). Потенциально возможен также вариант подключения внешних устройств к системной шине напрямую, без использования МВВ, но против него можно выдвинуть сразу несколько аргументов. Во-первых, в этом случае ЦП пришлось бы оснащать универсальными схемами для управления любым ВУ. При большом разнообразии внешних устройств, имеющих к тому же различные принципы действия, такие схемы оказываются чересчур сложными и избыточными. Во-вторых, пересылка данных при вводе и выводе происходит значительно медленнее, чем при обмене между ЦП и памятью, и было бы невыгодно задействовать для обмена информацией с ВУ высокоскоростную системную шину. И, наконец, в ВУ часто используются иные форматы данных и длина слова, чем в ВМ, к которым они подключены.

Адресное пространство системы ввода/вывода

Как и обращение к памяти, операции ввода/вывода также предполагают наличие некоторой системы адресации, позволяющей выбрать один из модулей СВВ, а также одно из подключенных к нему внешних устройств. Адрес модуля и ВУ является

составной частью соответствующей команды, в то время как расположение данных на внешнем устройстве определяется пересылаемой на ВУ информацией.

Адресное пространство ввода/вывода может быть совмещено с адресным пространством памяти или быть выделенным.

*При совмещении адресного пространства для адреса*ц и и *модулей* ввода/вывода отводится определенная область адресов (рис. 8.2). Обычно все операции с модулем ввода/вывода осуществляются с использованием входящих в него внутренних регистров: управления, состояния, данных. Фактически процедура ввода/вывода сводится к записи информации в одни регистры МВБ и считыванию ее из других регистров. Это позволяет рассматривать регистры МВБ как ячейки основной памяти и работать с ними с помощью обычных команд обращения к памяти, при этом в системе команд ВМ вообще могут отсутствовать специальные команды ввода в вывода. Так, модификацию регистров МВБ можно производить непосредственно с помощью арифметических и логических команд. Адреса регистрам МВВ назначаются в области адресного пространства памяти, отведенной под систему ввода/вывода.

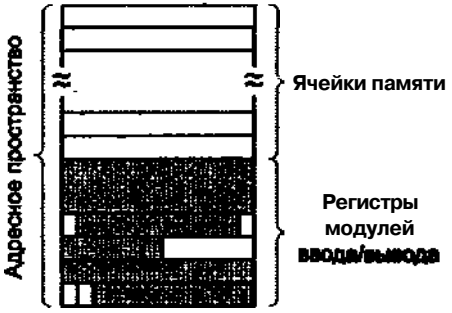


Рис. 8.2. Распределение совмещенного адресного пространства

Такой подход представляется вполне оправданным, если учесть, что ввод/вывод обычно составляет малую часть всех операций, выполняемых вычислительной машиной, чаще всего не более 1% от общего числа команд в программе.

Реализация концепции совмещенного адресного пространства в ВМ с кэш-памятью и виртуальной адресацией сопряжена с определенными проблемами. В частности, усложняется отображение виртуального адреса устройства ввода/вывода на физическое ВУ. Сложности также возникают и с кэшированием регистров МВВ.

Сформулируем преимущества и недостатки совмещенного адресного пространства.

Достоинства совмещенного адресного пространства:

- расширение набора команд для обращения к внешним устройствам, что позволяет сократить длину программы и повысить быстродействие;
- значительное увеличение количества подключаемых внешних устройств;
- возможность внепроцессорного обмена данными между внешними устройствами, если в системе команд есть команды пересылки между ячейками памяти;
- возможность обмена информацией не только с аккумулятором, но и с любым регистром центрального процессора.

Недостатки совмещенного адресного пространства:

- сокращение области адресного пространства памяти;
- усложнение декодирующих схем адресов в СВВ;
- трудности распознавания операций передачи информации при вводе/выводе среди других операций. Сложности в чтении и отладке программы, в которой простые команды вызывают выполнение сложных операций ввода/вывода;
- трудности при построении СВВ на простых модулях ввода/вывода: сигналы управления не смогут координировать сложную процедуру ввода/вывода. Поэтому МВБ часто должны генерировать дополнительные сигналы под управлением программы.

Совмещенное адресное пространство используется в вычислительных машинах MIPS и SPARC.

В случае *выделенного адресного пространства* для обращения к модулям ввода/вывода применяются специальные команды и отдельная система адресов. Это позволяет разделить шины для работы с памятью и шины ввода/вывода, что дает возможность совмещать во времени обмен с памятью и ввод/вывод. Кроме того, адресное пространство памяти может быть использовано по прямому назначению в полном объеме. В вычислительных машинах фирмы IBM и микроЭВМ на базе процессоров фирмы Intel система ввода/вывода, как правило, организуется в со-

Достоинства выделенного адресного пространства:

- адрес внешнего устройства в команде ввода/вывода может быть коротким. В большинстве СВВ количество внешних устройств намного меньше количества ячеек памяти. Короткий адрес ВУ подразумевает такие же короткие команды ввода/вывода и простые дешифраторы;
- программы становятся более наглядными, так как операции ввода/вывода выполняются с помощью специальных команд;
- разработка СВВ может проводиться отдельно от разработки памяти.

Недостатки выделенного адресного пространства:

- ввод/вывод производится только через аккумулятор центрального процессо- Для передачи информации от ВУ в РОН, если аккумулятор занят, требуется выполнение четырех команд (сохранение содержимого аккумулятора, ввод из ВУ, пересылка из аккумулятора в РОН, восстановление содержимого аккумулятора);
- перед обработкой содержимого ВУ это содержимое нужно переслать в ЦП.

Внешние устройства

Связь ВМ с внешним миром осуществляется с помощью самых разнообразных внешних устройств. Каждое ВУ подключается к МВБ посредством индивидуальной шины. Интерфейс, по которому организуется такое взаимодействие МВБ и ВУ, часто называют *малым*. Индивидуальная шина обеспечивает обмен данными и уп-

равляющими сигналами, а также информацией о состоянии участников обмена. Внешнее устройство, подключенное к МВБ, обычно называют *периферийным устройством* (ПУ). Все множество ПУ можно свести к трем категориям [200]:

- для общения с пользователем;
- для общения с ВМ;
- для связи с удаленными устройствами.

Примерами первой группы служат видеотерминалы и принтеры. Ко второй группе причисляются внешние запоминающие устройства (магнитные и оптические диски, магнитные ленты и т, п.), датчики и исполнительные механизмы. Отметим двойственную роль внешних ЗУ, которые, с одной стороны, представляют собой часть памяти ВМ, а с другой - являются внешними устройствами. Наконец, устройства третьей категории позволяют ВМ обмениваться информацией с удаленными объектами, которые могут относиться к двум первым группам. В роли удаленных объектов могут выступать также другие ВМ.

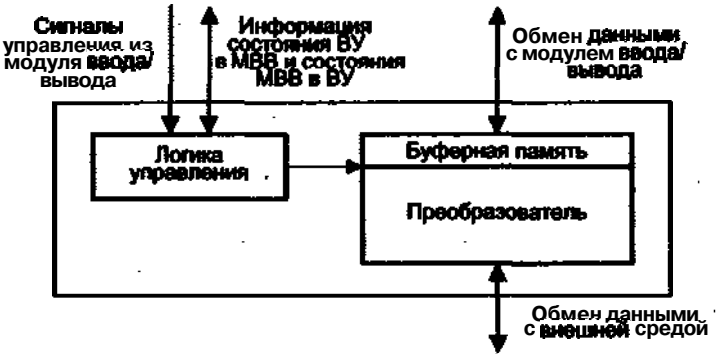


Рис. 8.3. Структура внешнего устройства

Обобщенная структура ВУ показана на рис. 8.3. Интерфейс с МВВ реализуется в виде сигналов управления, состояния и данных. *Данные* представлены совокупностью битов, которые должны быть переданы в модуль ввода/вывода или получены из него. *Сигналы управления* определяют функцию, которая должна быть выполнена внешним устройством. Это может быть стандартная для всех устройств функция — посылка данных в МВВ или получение данных из него, либо специфичная для данного типа ВУ функция, такая, например, как позиционирование головки магнитного диска или перемотка магнитной ленты. *Сигналы состояния* характеризуют текущее состояние устройства, в частности включено ли ВУ и готово ли оно к передаче данных.

Логика управления — это схемы, координирующие работу ВУ в соответствии с направлением передачи данных. Задачей *преобразователя* является трансформация информационных сигналов, имеющих самую различную физическую природу, в электрические сигналы, а также обратное преобразование. Обычно совместно с преобразователем используется *буферная память,* обеспечивающая временное хранение данных, пересылаемых между МВВ и ВУ.

Модули ввода/вывода

Функции модуля

Модуль ввода/вывода в составе вычислительной машины отвечает за управление одним или несколькими ВУ и за обмен данными между этими устройствами с одной стороны, и основной памятью или регистрами ЦП - с другой. Основные функции МВБ можно сформулировать следующим образом:

- локализация данных;
- управление и синхронизация;
- обмен информацией;
- буферизация данных;
- обнаружение ошибок.

Локализация данных

Под локализацией данных будем понимать возможность обращения к одному из ВУ, а также адресации данных на нем.

Адрес ВУ обычно содержится в адресной части команд ввода/вывода. Как уже отмечалось, в состав СВВ могут входить несколько модулей ввода/вывода. Каждому модулю назначается определенный диапазон адресов, независимо от того, является ли пространство адресов совмещенным или раздельным. Старшие разряды в адресах диапазона, выделенного модулю, обычно одинаковы и обеспечивают выбор одного из МВБ в рамках системы ввода/вывода. Младшие разряды адреса представляют собой уникальные адреса регистров данного модуля или подключенных к нему ВУ.

Одной из функций МВВ является проверка вхождения поступившего по шине адреса в выделенный данному модулю диапазон адресов. При положительном ответе модуль должен обеспечить дешифровку поступившего адреса и перенаправление информации к адресуемому объекту или от него.

Для простейших внешних устройств (клавиатура, принтер и т. п.) адрес ВУ однозначно определяет и расположение данных на этом устройстве. Для более сложных ВУ, таких как внешние запоминающие устройства, информация о местонахождении данных требует детализации. Так, для ЗУ на магнитной ленте необходимо указать номер записи, а для магнитного диска — номер цилиндра, номер сектора и т. п. Эта часть адресной информации передается в МВВ не по шине адреса, а в виде служебных сообщений, пересылаемых по шине данных. Обработка такой информации в модуле, естественно, сложнее, чем выбор нужного регистра или ВУ. В частности, она может требовать от МВВ организации процедуры поиска на носителе информации.

Управление и синхронизация

Функция управления и синхронизации заключается в том, что МВВ должен координировать перемещение данных между внутренними ресурсами ВМ и внешними устройствами. При разработке системы управления и синхронизации модуля ввода/вывода необходимо учитывать целый ряд факторов.

Прежде всего, нужно принимать во внимание, что ЦП может взаимодействовать одновременно с несколькими ВУ, причем быстродействие подключаемых к МВБ внешних устройств варьируется в очень широких пределах — от нескольких байтов в секунду в терминалах до десятков миллионов байтов в секунду при обмене с магнитными дисками. Если в системе используются шины, каждое взаимодействие между ЦП и **МВБ** включает в себя одну или несколько процедур арбитража.

В отличие от обмена с памятью процессы ввода/вывода и работа ЦП протекают не синхронно. Очередная порция информация может быть выдана на устройство вывода лишь тогда, когда это устройство готово их принять. Аналогично, ввод от устройства ввода допустим только в случае доступности информации на устройстве ввода. Несинхронный характер процессов ввода/вывода предполагает обмен сигналами, аналогичный процедуре «рукопожатия» (handshake), описанной в главе 4. Для двухпроводной системы синхронизации эта процедура состоит из четырех шагов, которые применительно к операции вывода можно описать следующим образом:

1. Центральный процессор с помощью сигнала ДД - 1 (данные достоверны) изве-

2. Приняв данные, устройство вывода сообщает процессору об их получении сигналом ДП = 1 (данные приняты).
3. Получив подтверждение, ЦП обнуляет сигнал ДД и снимает данные с шины, после чего может выставить на шину новые данные.
4. Обнаружив, что ДД = 0, устройство вывода, в свою очередь, устанавливает в нулевое состояние сигнал ДП , после чего оно готово для обработки принятых данных все время до получения очередного сигнала ДД = 1.

Описанную процедуру иллюстрирует рис. 8.4 (в скобках указаны номера шагов).

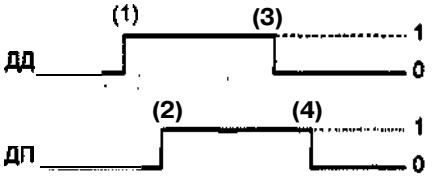


Рис. 8.4. временная диаграмма процедуры «рукопожатия»

Таким образом, модуль ввода/вывода обязан снабдить центральный процессор информацией о собственной готовности к обмену, а также о готовности подключенных к модулю ВУ. Помимо этого, процессор должен обладать оперативными сведениями и об иных происходящих в СВВ событиях.

Обмен информацией

Основной функцией **МВВ** является обеспечение обмена информацией. Со стороны «большого» интерфейса - это обмен с ЦП, а со стороны «малого» интерфейса - обмен с ВУ. В таком плане требования к МВВ непосредственно проистекают

из типовой последовательности операций, выполняемых процессором при вводе/выводе.

1. Выбор требуемого внешнего устройства.
2. Определение состояния МВБ и ВУ.
3. Выдача указания модулю ввода/вывода на подключение нужного ВУ к процессору.
4. Получение от МВБ подтверждения о подключении затребованного ВУ к процессору.
5. Распознавание сигнала готовности устройства к передаче очередной порции информации.
6. Прием (передача) порции информации.
7. Циклическое повторение двух предшествующих пунктов до завершения передачи информации в полном объеме.
8. Логическое отсоединение ВУ от процессора.

С учетом описанной процедуры функция *обмена информацией* с ЦП включает в себя:

- *дешифровку команды:* МВБ получает команды из ЦП в виде сигналов на шине управления;
- *пересылку данных* между МВБ и ЦП по шине данных;.
- *извещение о состоянии:* из-за того, что ВУ — медленные устройства, важно знать состояние модуля ввода/вывода. Так, в момент получения запроса на пересылку данных в центральный процессор МВБ может быть не готов выполнить эту пересылку, поскольку еще не завершил предыдущую команду. Этот факт должен быть сообщен процессору с помощью соответствующего сигнала. Возможны также сигналы, уведомляющие о возникших ошибках;
- *распознавание адреса;* МВБ обязан распознавать адрес каждого ВУ, которым он управляет.

Наряду с обеспечением обмена с процессором МВВ должен выполнять функцию обмена информацией с ВУ, Такой обмен также включает в себя передачу данных, команд и информации о состоянии.

Буферизация

Важной задачей модуля ввода/вывода является буферизация данных, необходимость которой иллюстрирует табл. 8.1 [120].

Таблица 8 . 1 . Примеры устройств ввода/вывода, упорядоченные по режиму работы, субъекту и скорости передачи данных

Устройств	Режим работы	Партнер	Скорость передачи данных, Кбайт/с
Клавиатура	Ввод	Человек	**0.01**
Мышь	Ввод	Человек	**0.02**

Устройство	Режим работы	Партнер	Скорость передачи данных, Кбайт/с
Сканер	Ввод	Человек	200
Строчный принтер	Вывод	Человек	1
Лазерный принтер	Вывод	Человек	100
Графический дисплей	Вывод	Человек	30 000
Локальная сеть	Ввод/вывод	ВМ	200
Гибкий диск	Память	ВМ	50
Оптический диск	Память	ВМ	500
Магнитный диск	Память	ВМ	2000

Несмотря на различия в скорости обмена информацией для разных ВУ, все они в этом плане значительно отстают от ЦП и памяти. Такое различие компенсируется за счет буферизации. При выводе информации на ВУ данные пересылаются из основной памяти в МВБ с большой скоростью. В модуле эти данные буферизируются и затем направляются в ВУ со скоростью, свойственной последнему. При вводе из ВУ данные буферизируются так, чтобы не заставлять память работать в режиме медленной передачи. Таким образом, МВБ должен обладать способнос-

Обнаружение ошибок

Еще одной из важнейших функций МВБ является обнаружение ошибок, возникающих в процессе ввода/вывода. Центральный процессор следует оповещать о каждом случае обнаружения ошибки. Причинами возникновения последних бывают самые разнообразные факторы, которые в первом приближении можно свести к следующим группам:

в воздействие внешней среды;

- старение элементной базы;
- системное программное обеспечение;
- пользовательское программное обеспечение.

Из наиболее "активных", факторов окружения ВМ следует выделить:

- загрязнение и влагу;
- повышенную или пониженную температуру окружающей среды;
-электромагнитное облучение;
- скачки напряжения питания.

Степень влияния каждого из этих факторов зависит от типа и конструкции МВБ и ВУ, Так, к загрязнению наиболее чувствительны оптические и механические элементы ВУ, в то время как работа электронных компонентов СВВ в большей степени зависит от температуры внешней среды, электромагнитного воздействия и стабильности питающего напряжения.

Фактор старения характерен как для механических, так и для электронных элементов, СВВ. В механических элементах он выражается в виде износа, следствием чего может быть неточное позиционирование головок считывания/записи на внешних запоминающих устройствах или неправильная подача бумаги в принтерах. Старение электронных элементов обычно выражается в изменении электрических параметров схем, приводящем к нарушению управления и синхронизации. Так, отклонения в параметрах электронных компонентов в состоянии вызвать недопустимый "перекос" сигналов, передаваемых между ЦП и МВБ или внутри МВБ.

Источником ошибок может стать и несовершенство системного программного обеспечения (ПО):

Ш непредвиденные последовательности команд или кодовые комбинации;

- некорректное распределение памяти;

- недостаточный размер буфера ввода/вывода;

* недостаточно продуманные и оттестированные комбинации системных модулей.

Среди ошибок, порождаемых пользовательским ПО, наиболее частыми являются:

Я нарушение последовательности выполнения программы;

- некорректные процедуры.

Вероятность возникновения ошибки внутри процессора для современных ЦП оценивается величиной порядка 10^{-18}, в то время как для остальных составляющих ВМ она лежит в диапазоне 10^{-8}–10^{-12}.

Способы обнаружения и исправления ошибок ввода/вывода практически не отличаются от рассмотренных в главе 5.

Структура модуля

Структура МВВ в значительной мере зависит от числа и сложности внешних устройств, которыми он управляет, однако в самом общем виде такой модуль можно представить в форме, показанной на рис. 8,5.

Связь модуля ввода/вывода с ядром ВМ осуществляется посредством системной или специализированной шины. С этой стороны в МВВ реализуется так называемый "большой" интерфейс. Большие различия в архитектуре систем команд и шин ВМ являются причиной того, что со стороны "большого" интерфейса модули ввода/вывода достаточно трудно унифицировать, и часто МВВ, созданные для одних ВМ, не могут быть использованы в других. Тем не менее в структурном плане они достаточно схожи.

Данные, передаваемые в модуль и из него, буферизируются в регистре данных. Буферизация позволяет компенсировать различие в быстродействии ядра ВМ и внешних устройств. Разрядность регистра, как правило, совпадает с шириной шины данных со стороны «большого» интерфейса (2, 4 или 8 байт). В свою очередь, большинство ВУ ориентировано на побайтовый обмен информацией. Побайтовая пересылка информации по «широкой» системной шине - крайне неэффективное решение, поэтому со стороны «малого» интерфейса регистр данных часто дополняют узлом упаковки/распаковки (на схеме не показан). Этот узел при вводе обеспечивает последовательное побайтовое заполнение регистра данных (упаковку),

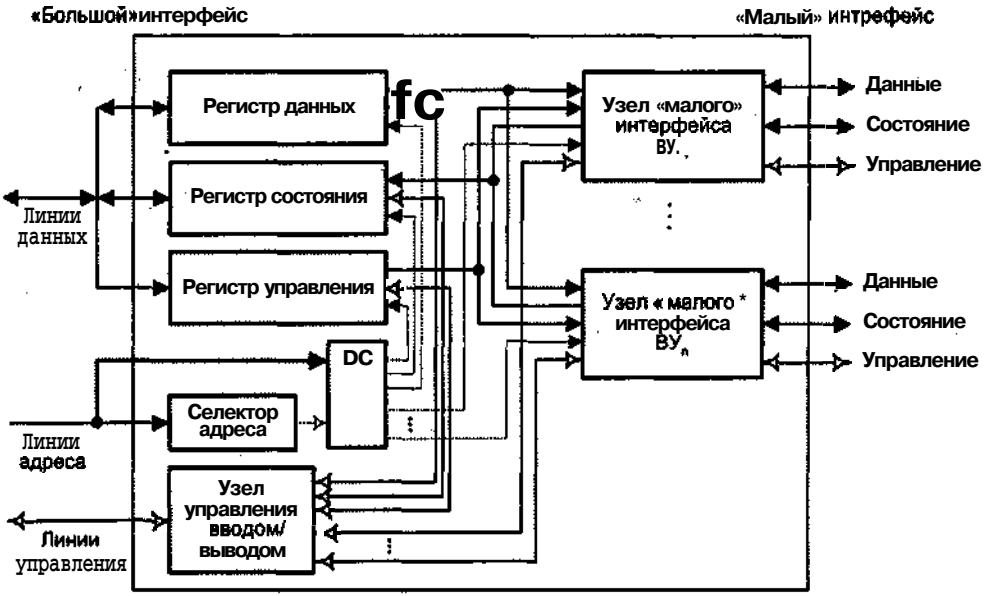


Рис. 8.5. Структура модуля ввода/вывода

а при выводе — последовательную побайтовую выдачу содержимого регистра на ВУ (распаковку). В результате при обмене данными через «большой» интерфейс занята вся ширина шины данных. В МВБ, рассчитанных на работу с большим числом ВУ, могут входить несколько регистров данных, что позволяет независимо хранить текущие данные каждого из внешних устройств.

Помимо регистра данных в составе МВБ имеются также регистр управления и регистр состояния (либо совмещенный регистр управления/состояния).

В *регистре управления* (РУ) фиксируются поступившие из ЦП команды управления модулем или подключенными к нему внешними устройствами. Отдельные разряды регистра могут представлять такие команды, как очистка регистров МВБ, сброс ВУ, начало чтения, начало записи и т. п. В сложных МВВ присутствует несколько регистров управления, например регистр управляющих сигналов для модуля в целом и отдельные РУ для каждого из ВУ.

Регистр состояния (РС) служит для хранения битов состояния МВВ и подключенных к нему ВУ. Содержимое определенного разряда регистра может характеризовать, например, готовность устройства ввода к приему очередной порции данных, занятость устройства вывода или нахождение ВУ в автономном режиме (offline). В МВВ не исключается наличие и более одного регистра состояния.

Процедура ввода/вывода предполагает возможность работы с каждым регистром МВВ или внешним устройством по отдельности. Такая возможность обеспечивается системой адресации. Каждому модулю в адресном пространстве ввода/вывода (совмещенном или раздельном) выделяется уникальный набор адресов, количество адресов в котором зависит от числа адресуемых элементов. Поступивший из ЦП адрес с помощью селектора адреса проверяется на принадлежность к диапазону, выделенному данному МВВ. В случае подтверждения дешифратор

DC выполняет раскодирование адреса, разрешая работу с соответствующим регистром модуля или ВУ.

Узел управления вводом/выводом по сути играет роль местного устройства управления МВВ. На него возлагаются две задачи: обеспечение взаимодействия с ЦП и координация работы всех составляющих МВВ. Связь с ЦП реализуется посредством линий управления, по которым из ЦП в модуль поступают сигналы, служащие для синхронизации операций ввода и вывода. В обратном направлении передаются сигналы, информирующие о происходящих в модуле событиях, например сигналы прерывания. Часть линий управления может задействоваться модулем для арбитража. Вторая функция узла управления реализуется с помощью внутренних сигналов управления.

Со стороны «малого»интерфейса МВВ обеспечивает подключение внешних устройств и взаимодействие с ними. Эта часть МВВ более унифицирована, поскольку внешние устройства всегда подгоняются под один из стандартных протоколов. Каждое из внешних устройств «обслуживается»- своим узлом «малого» интерфейса, который реализует принятый для данного ВУ стандартный протокол взаимодействия.

При управлении широким спектром ВУ модуль должен по возможности освобождать ЦП от знания деталей конкретных ВУ, так чтобы ЦП мог управлять любым устройством с помощью простых команд чтения и записи. МВВ при этом берет на себя задачи синхронизации, согласования форматов данных и т. п.

Модуль ввода/вывода, который берет на себя детальное управление ВУ и общается с ЦП только с помощью команд высокого уровня, часто называют *каналом ввода/вывода или процессором ввода/вывода. Наиболее прим*итивный *МВВ*, требующий детального управления со стороны ЦП, называют *контроллером ввода/вывода или контроллером устройства*. Как правило,' контроллеры ввода/вывода типичны для микроЭВМ, а каналы ввода/вывода - для универсальных ВМ.

Методы управления вводом/выводом

В ВМ находят применение три способа организации ввода/вывода (В/ВЫВ):

- программно управляемый ввод/вывод;
- ввод/вывод по прерываниям;
- прямой доступ к памяти.

При *программно управляемом вводе/выводе* все связанные с этим действия происходят по инициативе центрального процессора и под его полным контролем. ЦП выполняет программу, которая обеспечивает прямое управление процессом ввода/вывода, включая проверку состояния устройства, выдачу команд ввода или вывода. Выдав в МВВ команду, центральный процессор должен ожидать завершения ее выполнения, и, поскольку ЦП работает быстрее, чем МВВ, это приводит к потере времени.

Ввод/вывод по прерываниям во многом совпадает с программно управляемым методом. Отличие состоит в том, что после выдачи команды ввода/вывода ЦП не должен циклически опрашивать МВВ для выяснения состояния устройства. Вместо этого процессор может продолжать выполнение других команд до тех пор, пока

не получит запрос прерывания от МВВ, извещающий о завершении выполнения ранее выданной команды В/ВЫВ. Как и при программно управляемом В/ВЫВ, ЦП отвечает за извлечение данных из памяти (при выводе) и запись данных в память (при вводе).

Повышение как скорости В/ВЫВ, так и эффективности использования ЦП обеспечивает третий способ В/ВЫВ - *прямой доступ к памяти* (ПДП). В этом режиме основная память и модуль ввода/вывода обмениваются информацией напрямую, минуя процессор.

Программно управляемый ввод/вывод

Наиболее простым методом управления вводом/выводом является *программно управляемый ввод/вывод, часто называемый также вводом/выводом с опросом*. Здесь ввод/вывод происходит под полным контролем центрального процессора и реализуется специальной процедурой ввода/вывода. В этой процедуре ЦП с помощью команды ввода/вывода сообщает модулю ввода/вывода, а через него и внешнему устройству о предстоящей операции. Адрес модуля и ВУ, к которому производится обращение, указывается в адресной части команды ввода или вывода. Модуль исполняет затребованное действие, после чего устанавливает в единицу соответствующий бит в своем регистре состояния. Ничего другого, чтобы уведомить ЦП, модуль не предпринимает. Следовательно, для определения момента завершения операции или пересылки очередного элемента блока данных процессор должен периодически опрашивать и анализировать содержимое регистра состояния МВВ.

Иллюстрация процедуры программно управляемого ввода блока данных с устройства ввода приведена на рис. 8.6. Данные читаются пословно. Для каждого читаемого слова ЦП должен оставаться в цикле проверки, пока не определит, что слово находится в регистре данных МВВ, то есть доступно для считывания.

Процедура начинается с выдачи процессором команды ввода, в которой указан адрес конкретного МВВ и конкретного ВУ. Существуют четыре типа команд В/ВЫВ, которые может получить МВВ: управление, проверка, чтение и запись.

Команды управления используются для активизации ВУ и указания требуемой операции. Например, в устройство памяти на магнитной ленте может быть выдана команда перемотки или продвижения на одну запись. Для каждого типа ВУ характерны специфичные для него команды управления.

Команда проверки применяется для проверки различных ситуаций- возникающих в МВВ и ВУ в процессе ввода/вывода. С помощью таких команд ЦП способен выяснить, включено ли ВУ, готово ли оно к работе, завершена ли последняя операция ввода/вывода и не возникли ли в ходе ее выполнения какие-либо ошибки. Действие команды сводится к установке или сбросу соответствующих разрядов регистра состояния МВВ.

Команда чтения побуждает модуль получить элемент данных из ВУ и занести его в регистр данных (РД). ЦП может получить этот элемент данных, запросив МВВ поместить его на шину данных.

Команда записи заставляет модуль принять элемент данных (байт или слово) с шины данных и переслать его в РД с последующей передачей в ВУ.

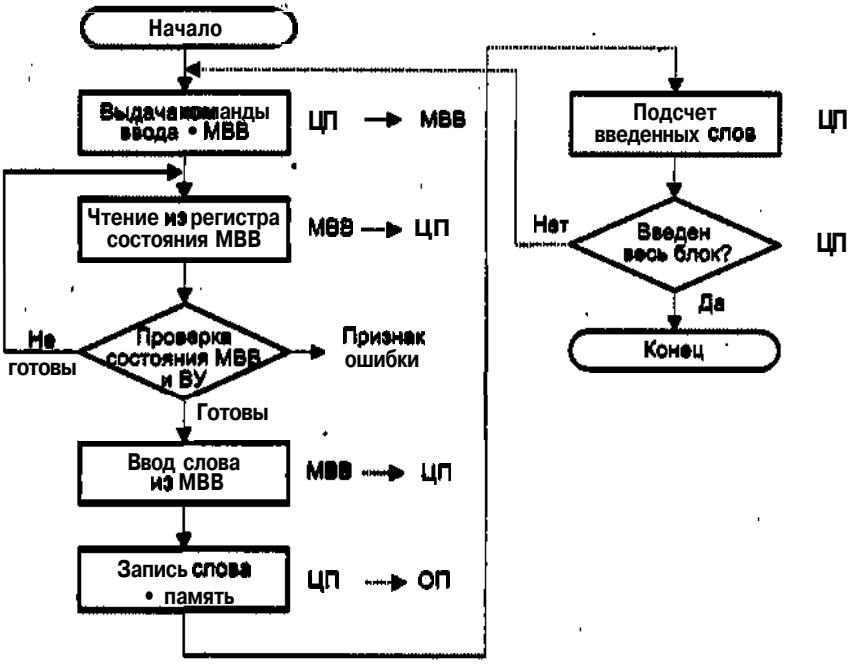


Рис. 8.6. Программно управляемый ввод данных

Если к МВВ подключено несколько ВУ, то в процедуре ввода/вывода нужно производить циклический опрос всех устройств, с которыми в данный момент прох изводятся операции В/ВЫВ.

Из блок-схемы (см. рис. 8.6) явно виден основной недостаток программно упх равляемого В/ВЫВ - неэффективное использование процессора из-за ожидания готовности очередной порции информации, в течение которого никаких иных пох лезных действий ЦП не выполняет. Кроме того, пересылка даже одного слова трех бует выполнения нескольких команд. ЦП должен тратить время на анализ битов состояния МВВ, запись в МВВ битов управления, чтение или запись данных со скоростью, определяемой внешним устройством. Все это также отрицательно ских зывается на эффективности ввода/вывода.

Главным аргументом в пользу программно управляемого ввода/вывода являх ется простота МВВ, поскольку основные функции по управлению В/ВЫВ берет на себя процессор. При одновременной работе с несколькими ВУ приоритет устройств легко изменить программными средствами (последовательностью опроса). Наконец, подключение к СВВ новых внешних устройств или отключение ранее подключенных также реализуется без особых сложностей.

Ввод/вывод по прерываниям

Как уже отмечалось, основным недостатком программно управляемого В/ВЫВ являются простои процессора в ожидании, пока модуль ввода/вывода выполнит очередную операцию. Альтернативой может быть вариант, когда ЦП выдает ко-

манду В/ВЫВ, а затем продолжает делать другую полезную работу. Когда ВУ готово к обмену данными, оно через МВБ извещает об этом процессор с помощью запроса на прерывание. ЦП осуществляет передачу очередного элемента данных, после чего возобновляет выполнение прерванной программы.

Обсудим процесс ввода блока данных с использованием В/БЫВ по прерываниям (рис. 8,7)." Оставим без внимания такие подробности, как сохранение и восстановления контекста, действия, выполняемые при завершении пересылки блока данных, а также в случае возникновения ошибок.

Процедура ввода блока данных по прерываниям реализуется следующим образом. ЦП выдает команду чтения, а затем продолжает выполнение других заданий, например другой программы. Получив команду, МВБ приступает к вводу элемента данных с ВУ. Когда считанное слово оказывается в регистре данных модуля, МВВ формирует на управляющей линии сигнал прерывания ЦП. Выставив запрос, МВБ помещает введенную информацию на шину данных, после чего он готов к следующей операции В/ВЫВ. ЦП в конце каждого цикла команды проверяет наличие запросов прерывания. Когда от МВВ приходит такой сигнал, ЦП сохраняет контекст текущей программы и обрабатывает прерывание. В рассматриваемом случае ЦП читает слово из модуля, записывает его в память и выдает модулю команду на считывание очередного слова. Далее ЦП восстанавливает контекст прерванной программы и возобновляет ее выполнение.

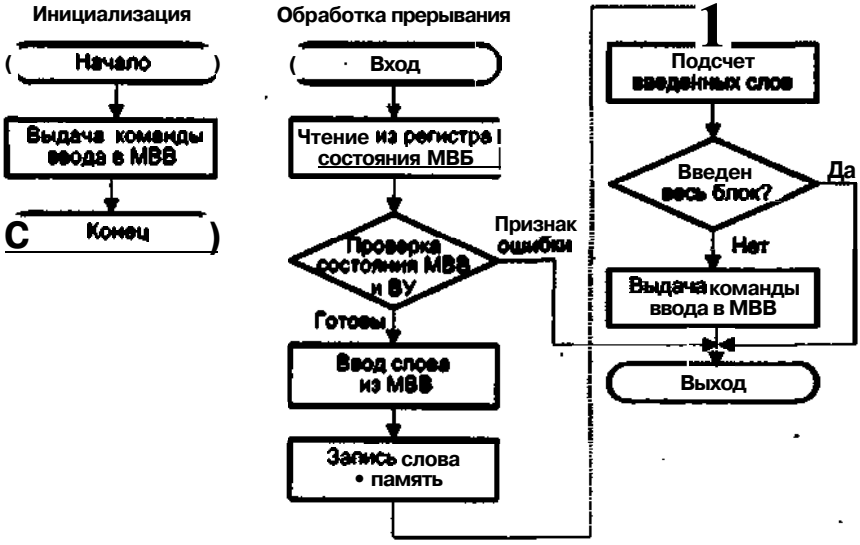


Рис. 8.7. Ввод данных по прерыванию

Этот метод эффективнее программно управляемого В/ВЫВ, поскольку устраняет ненужные ожидания, однако обработка прерывания занимает достаточно много времени ЦП. Кроме того, каждое слово, пересылаемое из памяти в модуль В/ВЫВ или в противоположном направлении, как и при программно управляемом В/ВЫВ, проходит через ЦП.

Реализация ввода/вывода по прерываниям

При реализации ввода/вывода по прерываниям необходимо дать ответы на два вопроса. Во-первых, определить, каким образом ЦП может выяснить, какой из МВБ и какое из подключенных к этому модулю внешних устройств выставили запрос. Во-вторых, при множественных прерываниях требуется решить, какое из них должно быть обслужено в первую очередь.

Сначала рассмотрим вопрос идентификации устройства. Здесь возможны три основных метода:

- множественные линии прерывания;
- программная идентификация;
- векторное прерывание.

Наиболее простой подход к решению проблемы определения источника запроса — *применение множественных линий прерывания* между ЦП и модулями ввода/вывода, хотя выделение слишком большого количества управляющих линий для этих целей нерационально. Более того, даже если присутствует несколько линий прерывания, желательно, чтобы каждая линия использовалась всеми МВБ, при этом для каждой линии действует один из двух остальных методов идентификации устройства.

При *программной идентификации,* обнаружив запрос прерывания, ЦП переходит к общей программе обработки прерывания, задачей которой является опрос всех МВБ с целью определения источника запроса. Для этого может быть выделена специальная командная линия опроса. ЦП помещает на адресную шину адрес опрашиваемого МВБ и формирует на этой линии сигнал опроса. Реакция модуля зависит от того, выставлял он запрос или нет. Возможен и иной вариант, когда каждый МВБ включает в себя адресуемый регистр состояния. Тогда ЦП считывает содержимое РС каждого модуля, после чего выясняет источник прерывания. Когда источник прерывания установлен, ЦП переходит к программе обработки прерывания, соответствующей этому источнику. Недостаток метода программной идентификации заключается в больших временных потерях.

Наиболее эффективную процедуру идентификации источника прерывания обеспечивают аппаратурные методы, в основе которых лежит идея *векторного прерывания*. В этом случае, получив подтверждение прерывания от процессора, выставившее запрос устройство выдает на шину данных специальное слово, называемое *вектором прерывания*. Слово содержит либо адрес МВБ, либо какой-нибудь другой уникальный идентификатор, который ЦП интерпретирует как указатель на соответствующую программу обработки прерывания. Такой подход устраняет необходимость в предварительных действиях с целью определения источника запроса прерывания. Реализуется он с помощью хранящейся в ОП *таблицы векторов прерывания* (рис. 8.8), где содержатся адреса программ обработки прерываний. Входом в таблицу служит вектор прерывания. Начальный адрес таблицы (база) обычно задается неявно, то есть под таблицу отводится вполне определенная область памяти.

Наиболее распространены два варианта векторной идентификации источника запроса прерывания: цепочечный опрос и арбитраж шины.

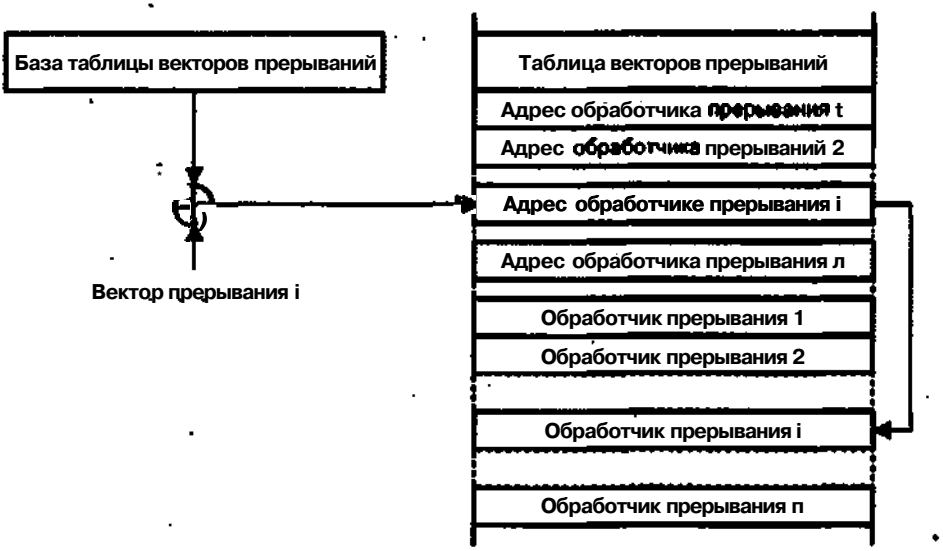


Рис. 8.8. Идентификация запроса с помощью вектора прерывания

При *цепочечном методе* для передачи запроса прерывания модули ввода/вывода совместно используют одну общую линию. Линия подтверждения прерывания последовательно проходит через все МВБ. Когда ЦП обнаруживает запрос прерывания, он посылает сигнал по линии подтверждения прерывания. Этот сигнал двих жется через цепочку модулей, пока не достигнет того, который выставил запрос. Запросивший модуль реагирует путем выдачи на шину данных своего вектора прерывания.

В варианте *арбитража шины* МВВ, прежде чем выставить запрос на линии запроса прерывания, должен получить право на управление шиной. Таким образом, в каждый момент времени активизировать линию запроса прерывания может только один из модулей. Когда ЦП обнаруживает прерывание, он отвечает по линии подтверждения. После этого запросивший модуль помещает на шину данных свой вектор прерывания.

Перечисленные методы служат не только для идентификации запросившего МВВ, но и для назначения приоритетор, когда прерывание запрашивают несколько устройств. При множественных линиях запроса ЦП начинает с линии, имеющей наивысший приоритет. В варианте программной идентификации приоритет модулей определяется очередностью их проверки. Для цепочечного метода приоритет модулей определяется порядком их следования в цепочке. Порядок задания приоритетов при арбитраже был рассмотрен ранее в главе 4.

Прямой доступ к памяти

Хотя ввод/вывод по прерываниям эффективнее программно управляемого, оба этих метода страдают двумя недостатками:

- темп передачи при вводе/выводе ограничен скоростью, с которой ЦП в состоя-- нии опросить и обслужить устройство;

* ЦП вовлечен в управление передачей, для каждой пересылки он должен выполнить определенное количество команд.

Когда пересылаются большие объемы данных, требуется более эффективный способ ввода/вывода - *прямой доступ к памяти* (ПДП). ПДП предполагает наличие на системной шине дополнительного модуля — *контроллера прямого доступа к памяти* (КПДП), способного брать на себя функции ЦП по управлению системной шиной и обеспечивать прямую пересылку информации между ОП и ВУ, без участия центрального процессора. В сущности, КПДП - это и есть модуль ввода/вывода, реализующий режим прямого доступа к памяти.

Если ЦП желает прочитать или записать блок данных, он прежде всего должен поместить в КПДП (рис. 8.9) информацию, характеризующую предстоящее действие. Этот процесс называется инициализацией КПДП и включает в себя занесение в контроллер следующих четырех параметров:

- вида запроса (чтение или запись);
- адреса устройства ввода/вывода;
- адреса начальной ячейки блока памяти, откуда будет извлекаться или куда будет вводиться информация;
- количества слов, подлежащих чтению или записи.

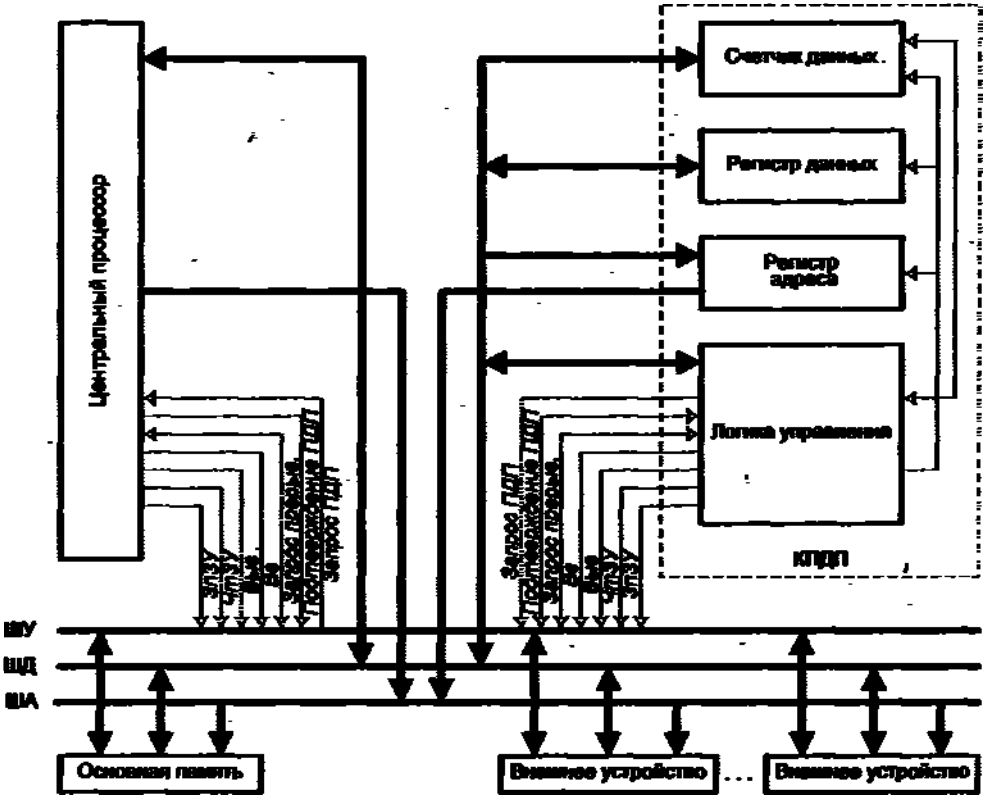


Рис. 8.9. Организация прямого доступа к памяти

Первый параметр определяет направление пересылки данных: из ОП в ВУ или наоборот. За исходную точку обычно принимается память, поэтому под чтением понимают считывание данных из ОП и выдачу их на устройство вывода, а под записью — прием данных из устройства ввода и запись в ОП. Вид запроса запоминается в схеме логики управления контроллера.

К КПДП обычно могут быть подключены несколько ВУ, а адрес УВВ конкретизирует, какое из них должно участвовать в предстоящем обмене данными. Этот адрес запоминается в логике управления КПДП.

Третий параметр - адрес начальной ячейки - хранится в регистре адреса (РА) контроллера. После передачи каждого слова содержимое РА автоматически увеличивается на единицу, то есть в нем формируется адрес следующей ячейки ОП.

Размер блока в словах заносится в счетчик данных (СД) контроллера. После передачи каждого слова содержимое СД автоматически уменьшается на единицу. Нулевое состояние СД свидетельствует о том, что пересылка блока данных завершена

После инициализации процесс пересылки информации может быть начат в любой момент. Инициаторами обмена вправе выступать как ЦП, так и ВУ. Устройство, желающее начать В/ВЫВ, извещает об этом контроллер подачей соответствующего сигнала. Получив такой сигнал, КПДП выдает в ЦП сигнал «Запрос ПДПк В ответ ЦП освобождает шины адреса и.данных, а также те линии шины управления, по которым передаются сигналы, управляющие операциями на шине адреса (ША) и шине данных (ШД). К таким, прежде всего, относятся линии ЧтЗУ, ЗпЗУ, Вы в, Вв и линия выдачи адреса на ША Далее ЦП отвечает контроллеру сигналом «Подтверждение ПДП», который для последнего означает, что ему делегированы права на управление системной шиной и можно приступать к пересылке данных.

Процесс пересылки каждого слова блока состоит из двух этапов.

При выполнении операции чтения (ОП -> ВУ) на первом этапе КПДП выставляет на шину адреса содержимое РА (адрес текущей ячейки ОП) и формирует сигнал ЧтЗУ. Считанное из ячейки ОП слово помещается на шину данных. На втором этапе КПДП выставляет на Ш А адрес устройства вывода и формирует сигнал Выв, который обеспечивает передачу слова с шины данных в ВУ.

При выполнении операции записи (ВУ -»ОП) КПДП сначала выдает на шину данных адрес устройства ввода и формирует сигнал Вв, по которому введенные данные поступают на шину данных. На втором этапе КПДП помещает на ША адрес ячейки ОП, куда должны быть занесены данные, и выдает сигнал ЗпЗУ. Этим сигналом информация с ШД записывается в ячейку ОП.

Как при чтении, так и при записи происходит буферизация пересылаемого слова в регистре данных (РД) контроллера. Это необходимо для компенсации различий в скорости работы ОП и ВУ, в силу чего сигналы Выв и Вв формируются контроллером лишь при получении от ВУ подтверждения о готовности. Буферизация сводится к тому, что после первого этапа слово с ШД заносится в РД, а перед вторым — возвращается,из РД на шину данных.

После пересылки каждого слова логика управления прибавляет единицу к содержимому РА (формирует адрес следующей ячейки ОП) и уменьшает на единицу содержимое СД (ведет подсчет переданных слов).

Когда пересылка завершена (при нулевом значении в СД), КПДП снимает сигнал «Запрос ПДП», в ответ на что ЦП снимает сигнал «Подтверждение ПДП» и вновь берет на себя управление системной шиной, то есть ЦП вовлечен в процесс ввода/вывода только в начале и конце передачи.

Эффективность ПДП зависит от того, каким образом реализовано распределение системной шины между ЦП и КПДП в процессе пересылки блока. Здесь может применяться один из трех режимов:

- блочная пересылка;
- пропуск цикла;
- прозрачный режим.

При *блочной пересылке* КПДП полностью захватывает системную шину с момента начала пересылки и до момента завершения передачи всего блока. На весь этот период ЦП не имеет доступа к шине.

В режиме *пропуска цикла* КПДП после передачи каждого слова на один цикл шины освобождает системную шину, предоставляя ее на это время процессору. Поскольку КПДП все равно должен ждать готовности ПУ, это позволяет ЦП эффективно распорядиться данным обстоятельством,

В *прозрачном режиме* КПДП имеет доступ к системной шине только в тех циклах, когда ЦП в ней не нуждается. Это обеспечивает наиболее эффективную работу процессора, но может существенно замедлять операцию пересылки блока данных. Здесь многое зависит от решаемой задачи, поскольку именно она определяет интенсивность использования шины процессором.

В отличие от обычного прерывания в пределах цикла команды имеется несколько точек, где КПДП вправе захватить шину (рис 8.10). Отметим, что это не прерывание: процессору не нужно запоминать контекст задачи.

Рис. 8.10, Точки возможного вмешательства в цикл команды при прямом доступе к памяти и при обычном прерывании

Механизм ПДП может быть реализован различными путями. Некоторые возможности показаны на рис. 8.11.

В первом примере (см. рис. 8.11, *а)* все ВУ совместно используют общую системную шину. КПДП работает как заменитель ЦП и обмен данными между памятью и ВУ через КПДП производит через программно управляемый ввод/вывод. Хотя этот вариант может быть достаточно дешевым, эффективность его невысока. Как и в случае программно управляемого ввода/вывода, осуществляемого процесором, каждая пересылка требует двух циклов шины.

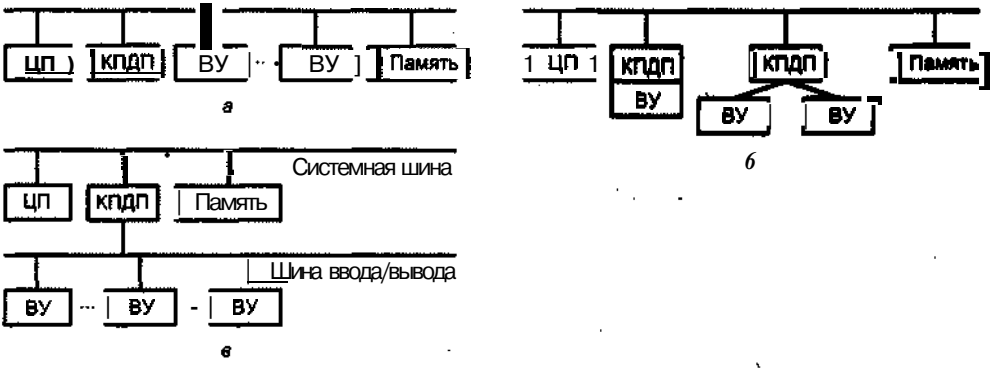


Рис. 8.11, возможные конфигурации систем прямого доступа к памяти

Число необходимых циклов шины может быть уменьшено при объединении функций КПДП и ВУ. Как видно из рис. 8.11,*б,* это означает, что между КПДП и одним или несколькими ВУ есть другой тракт, не включающий системную шину. Логика ПДП может быть частью ВУ, либо это может быть отдельный КПДП, управляющий одним или несколькими внешними устройствами. Допустим и еще один шаг в том же направлении (см. рис, 8.11, *в)* — соединение КПДП с ВУ посредством шины ввода/вывода. Это уменьшает число интерфейсов В/ВЫВ в КПДП и делает такую конфигурацию легко расширяемой. В двух последних вариантах системная шина задействуется КПДП только для обмена данными с памятью. Обмен данными между КПДП и ВУ реализуется минуя системную шину.

Каналы и процессоры ввода/вывода

По мере развития систем В/ВЫВ юс функции усложняются. Главная цель такого усложнения — максимальное высвобождение ЦП от управления процессами ввода/вывода. Некоторые пути решения этой задачи уже были рассмотрены. Следующими шагами в преодолении проблемы могут быть:

1. Расширение возможностей МВБ и предоставление ему прав процессора со специализированным набором команд, ориентированных на операции ввода/вывода ЦП дает указание такому процессору В/ВЫВ выполнить хранящуюся в памяти ВМ программу ввода/вывода. Процессор В/ВЫВ извлекает и исполняет команды этой программы без участия центрального процессора и прерывает ЦП только после завершения всей программы ввода/вывода.
2. Рассмотренному _в пункте 1 процессору ввода/вывода придается собственная локальная память, при этом возможно управление множеством устройств В/ВЫВ с минимальным привлечением ЦП.

В первом случае МВБ называют *каналом ввода/вывода* (КВВ), а во втором — *процессором ввода/вывода*. В принципе различие между каналом и процессором ввода/вывода достаточно условно, поэтому в дальнейшем будем пользоваться термином "канал".

Концепция системы ввода/вывода с КВВ характерна для больших универсальных вычислительных машин (мэйнфреймов), где проблема эффективной органи-

зации В/ВЫВ и максимального высвобождения центрального процессора в пользу его основной функции стоит наиболее остро. СВВ с каналами ввода/вывода была предложена и реализована в ВМ семейства IBM 360 и получила дальнейшее развитие в семействах IBM 370 и IBM 390.

В ВМ с каналами ввода/вывода центральный процессор практически не участвует в непосредственном управлении внешними устройствами, делегируя эту задачу специализированному процессору, входящему в состав КВВ. Все функции ЦП сводятся к запуску и остановке операций в КВВ, а также проверке состояния канала и подключенных к нему ВУ. Для этих целей ЦП использует лишь несколько (от 4 до 7) команд ввода/вывода. Например, в IBM 360 таких команд четыре:

- «Начать ввод/вывода;
- «Остановить ввод/вывод»;
- «Проверить ввод/вывод»;
- «Проверить канал».

КВВ реализует операции В/ВЫВ путем выполнения так называемой *канальной программы.* Канальные программы для каждого ВУ, с которым предполагается обмен информацией, описывают нужную последовательность операций ввода/вывода и хранятся в основной памяти ВМ. Роль команд в канальных программах выполняют *управляющие слова канала* (УСК), структура которых отличается от структуры обычной машинной команды. Типовое УСК содержит:

- *код операции,* определяющий для КВВ и ВУ тип операции: «Записать» (вывод информации из ОП в ВУ), «Прочитать» (ввод информации из ВУ в ОП), «Управление» (перемещение головок НМД, магнитной ленты и т. п.);
- *указатели* — дополнительные предписания, задающие более сложную последовательность операций В/ВЫВ, например при вводе пропускать отдельные записи или наоборот - с помощью одной команды вводить «разбросанный»- по ОП массив как единый;
- *адрес данных,* указывающий область памяти, используемую в операции ввода/вывода;
- *счетчик данных,* хранящий значение длины передаваемого блока данных.

Кроме того, в УСК может содержаться идентификатор ВУ и информация о его уровне приоритета, указания по действиям, которые должны быть произведены при возникновении ошибок и т. п.

Центральный процессор инициирует ввод/вывод путем инструктирования канала о необходимости выполнить канальную программу, находящуюся в ОП, и указания начального адреса этой программы в памяти ВМ. КВВ следует этим указаниям и управляет пересылкой данных. Отметим, что пересылка информации каналом ведется в режиме прямого доступа к памяти. ВУ взаимодействуют с каналом, получая от него приказы. Таким образом, в ВМ с КВВ управление вводом/выводом строится иерархическим образом. В операциях ввода/вывода участвуют три типа устройств:

- процессор (первый уровень управления);
- канал ввода/вывода (второй уровень);

- внешнее устройство (третий уровень).

Каждому типу устройств соответствует свой вид управляющей информации:

- процессору — команды ввода/вывода;
- каналу — управляющие слова канала;
- периферийному устройству - приказы.

Структура ВМ с канальной системой ввода/вывода показана на рис. 8.12.

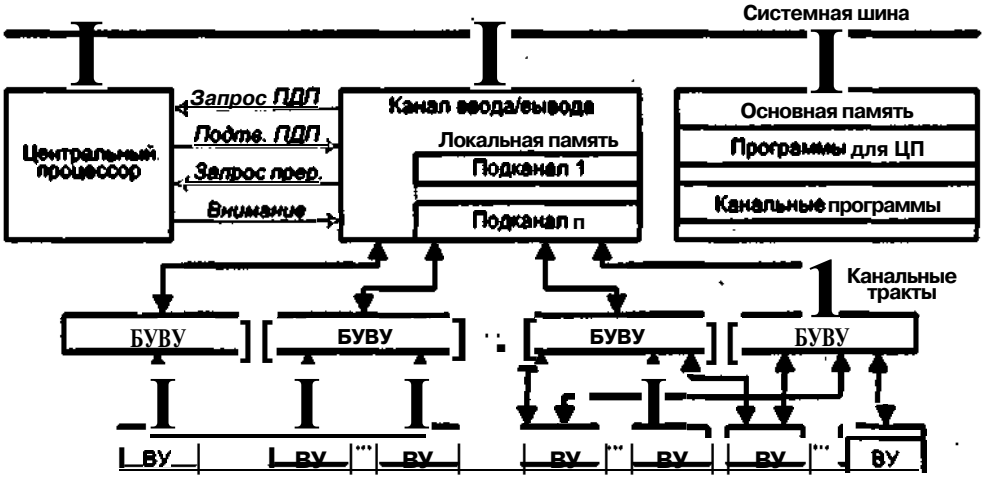


Рис. 8.12. ВМ с канальной системой ввода/вывода

Обмен информацией между КВВ и основной памятью осуществляется посредством системной шины ВМ. ВУ подключаются к каналу не непосредственно, а через блоки управления внешними устройствами (БУВУ). БУВУ принимает от канала приказы по управлению внешним устройством (чтение, запись, перемещение носителя или магнитной головки и т. п.) и преобразует их в сигналы управления, свойственные данному типу ВУ. Обычно один БУВУ может обслуживать несколько однотипных ВУ, но для подключения быстродействующих внешних устройств часто применяются индивидуальные блоки управления. В свою очередь, некоторые ВУ могут подключаться одновременно к нескольким БУВУ. Это позволяет воспользоваться свободным трактом другого БУВУ при занятости данного БУВУ обслуживанием одного из подключенных к нему ПУ. Физически БУ ВУ можетбыть самостоятельным устройством или интегрирован с ВУ или каналом.

Обмен информацией между БУВУ и КВВ обеспечивается так называемыми *канальными трактами*. Обычно каждое БУВУ связано с одним из канальных трактов, но возможно также подключение блока управления сразу к нескольким трактам, что дает возможность избежать нежелательных задержек при занятости одного из них.

В пределах канала ввода/вывода считается, что каждое ВУ подключено к своему *подканалу*. Подканалы имеют свои уникальные логические номера, с помощью которых канальная программа адресуется к конкретному ВУ. Физически подканал реализуется в виде участка памяти, в котором хранятся параметры операции

ввода/вывода, выполняемой данным ВУ: текущие значения адреса и счетчика данных, код и указатели операции ввода/вывода, адрес следующего УСК и др. Для хранения этих параметров обычно используется локальная память канала.

Обмен информацией между ВУ и ОП, как уже упоминалось, реализуется в режиме прямого доступа к памяти, при этом для взаимодействия ЦП и канала задействованы сигналы «Запрос ПДП» и «Подтверждение ПДП».

Чтобы известить ЦП об окончании текущей канальной программы или об ошибках, возникших при ее выполнении, КВВ выдает в ЦП сигнал «Запрос прерывания». В свою очередь, ЦП может привлечь внимание канала сигналом «Внимание».

Способ организации взаимодействия ВУ с каналом определяется соотношением быстродействия ОП и ВУ. По этому признаку ВУ образуют две группы: быстродействующие (накопители на магнитных дисках (НМД), накопители на магнитных лентах (НМЛ)) со скоростью приема и выдачи информации около 1 Мбайт/с и медленнодействующие (дисплеи, печатающие устройства и др.) со скоростями порядка 1 Кбайт/с и менее. Быстродействие основной памяти обычно значительно выше. С учетом производительности ВУ в КВВ реализуются два режима работы: *мультиплексный* (режим разделения времени) *и монопольный*.

В *мультиплексном режиме* несколько внешних устройств разделяют канал во времени, при этом каждое из параллельно работающих с каналом ВУ связывается с КВВ на короткие промежутки времени только после того, как ВУ будет готово к приему или выдаче очередной порции информации (байта, группы байтов и т. д.). Такая схема принята в *мультиплексном канале ввода/вывода*. Если в течение сеанса связи пересылается один байт или несколько байтов, образующих одно машинное слово, канал называется *байт-мультиплексным*. Канал, в котором в пределах сеанса связи пересылка данных выполняется поблочно, носит название *блок-мультиплексного*.

В *монопольном режиме* после установления связи между каналом и ВУ последнее монополизирует канал на все время до завершения инициированной процессором канальной программы и всех предусмотренных этой программой пересылок данных между ВУ и ОП. На все время выполнения канальной программы канал оказывается недоступным для других ВУ. Данную процедуру обеспечивает *селекторный канал ввода/вывода*. Отметим, что в блок-мультиплексном канале в рамках сеанса связи пересылка блока осуществляется в монопольном режиме.

Канальная подсистема

В последовавших за IBM 360 семействах универсальных ВМ семейства IBM 370 и особенно IBM 390 концепция системы ввода/вывода на базе каналов получила дальнейшее развитие и вылилась в так называемую *канальную подсистему ввода/вывода* (КПВВ). Главная идея заключается в интегрировании отдельных КВВ в единый специализированный процессор ввода/вывода с большим числом канальных трактов и подканалов. Блоки управления внешними устройствами обычно подключаются к нескольким канальным трактам, что позволяет динамически менять путь пересылки информации с учетом текущей их загруженности. Кроме того, разные канальные тракты могут обладать различной пропускной способностью, и при выборе трактов для подключения определенных ВУ может быть учтено их быстродействие.

Одной из наиболее совершенных канальных подсистем обладают ВМ семейства IBM 390. В ней предусмотрено использование до 65 536 подканалов и до 256 канальных трактов. Реализованы два типа канальных трактов: параллельный и последовательный.

Параллельные канальные тракты по своим возможностям и принципу действия аналогичны рассмотренным ранее мультиплексному и селекторному каналам, но в отличие от них являются универсальными, то есть могут работать в байт-мультиплексном, блок-мультиплексном и селекторном режимах. Такие канальные тракты в КПВВ называют параллельными, поскольку они обеспечивают пересылку информации параллельным кодом.

Для работы с ВУ, соединенными с КПВВ волоконно-оптическими линиями, используются *последовательные канальные тракты,* реализующие протокол ESCON (Enterprise Systems Connection Architecture). Последовательный канальный тракт рассчитан на передачу информации только в последовательном коде и только в селекторном режиме. Для подключения блоков управления внешними устройствами (БУВУ) к ESCON-тракту служат специальные устройства, называемые ESCON-директорами. Каждое такое устройство может обеспечить одновременное подключение до 60 БУВУ и одновременную передачу информации от 30 из них со скоростью до 10 Мбайт/с.

Кроме того, в КПВВ предусмотрены специальные коммуникационные канальные тракты для подключения к сетям ВМ, модемам, другим системам.

В принципе основное преимущество КПВВ — динамическое перераспределение канальных трактов - в какой-то мере может быть реализовано и в рамках каждого отдельного канала. Однако объединение всех канальных ресурсов в единую канальную подсистему позволяет применить оптимальную стратегию динамического распределения и использования этих ресурсов и благодаря этому достичь качественно нового уровня эффективности системы ввода/вывода.

Контрольные вопросы

1. Поясните достоинства и недостатки трех вариантов подключения системы ввода/вывода к процессору ВМ.
2. Сформулируйте достоинства, недостатки и область применения двух способов организации адресного пространства ввода/вывода
3. Дайте развернутую характеристику структуры ВУ, отображая ее элементы в каждый из трех типов ВУ.
4. В чем состоит локализация данных, выполняемая модулем ввода/вывода?
5. Опишите содержание процедуры «рукопожатия» при выполнении операции ввода.
6. Конкретизируйте последовательность действий процессора при обмене информацией с жестким диском.
7. Выберите конкретную скорость работы ЦП. Рассчитайте емкость буферной памяти МВБ для обмена с клавиатурой, символьным принтером и оптическим диском.

8. Проведите маленькое исследование: спрогнозируйте вероятность возникновения ошибки мыши, лазерного принтера, оптического диска. Ответы обоснуйте.
9. Для конкретного ЦП определите структуры МВБ для мыши, клавиатуры, жесткого диска. Необходимость элементов структуры обоснуйте.
10. Сравните известные вам методы управления вводом/выводом по трем параметрам: достоинствам, недостаткам, области применения.
11. Поясните классификацию методов ввода/вывода по прерыванию.
12. Охарактеризуйте известные вам режимы прямого доступа к памяти, сформулируйте их достоинства и недостатки.
13. Опишите процесс вывода пяти слов и ввода семи слов при трех вариантах реализации ПДП.
14. Сравните ввод/вывод по прерыванию с вводом/выводом при ПДП. Для какого режима ПДП эти методы наиболее близки и почему?
15. Проведите сравнительный анализ контроллера ПДП и канала ввода/вывода. В чем их сходство? Чем они отличаются друг от друга?
16. Опишите процесс взаимодействия ЦП и КВВ. Какая при этом используется управляющая информация?
17. Опишите задачи посредника между КВВ и ВУ.

Глава 9

Основные направления в архитектуре процессоров

Ранее уже были рассмотрены основные составляющие центрального процессора. В данном разделе основное внимание уделено вопросам общей архитектуры процессоров как единого устройства и способам- повышения их производительности.

Конвейеризация вычислений

Совершенствование элементной базы уже не приводит к кардинальному росту производительности ВМ. Более перспективными в этом плане представляются архитектурные приемы, среди которых один из наиболее значимых — *конвейеризация.*

Для пояснения идеи конвейера сначала обратимся к рис. 9.1, а, где показан отдельный функциональный блок (ФБ). Исходные данные помещаются во входной регистр $Рг_{вх}$, обрабатываются в функциональном блоке, а результат обработки фиксируется в выходном регистре $Рг_{вых}$. Если максимальное время обработки в ФБ равно T_{max}, то новые данные могут быть занесены во входной регистр Рг не ранее, чем спустя T_{max}.

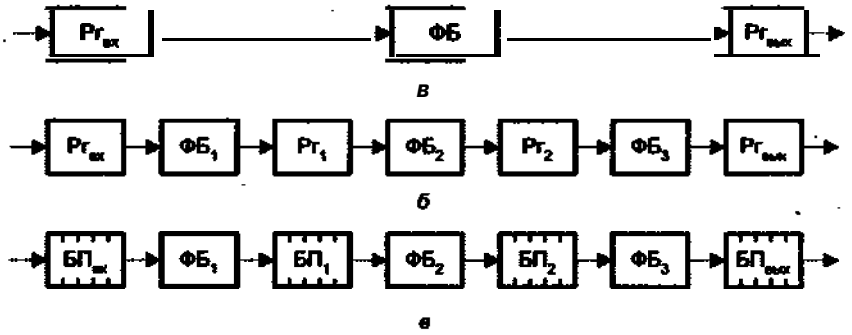


Рис. 9.1. Обработка информации: а - в одиночном блоке; б -в конвейере с регистрами; *в* - в конвейере с буферной памятью

Теперь распределим функции, выполняемые в функциональном блоке ФБ (см. рис. 9.1, а), между тремя последовательными независимыми блоками: $ФБ_1$, $ФБ_2$ и $ФБ_3$, причем так, чтобы максимальное время обработки в каждом ФБi было одинаковым и равнялось $T_{max}/3$. Между блоками разместим буферные регистры $Рг_i$ (рис. 9.1, *б),* предназначенные для хранения результата обработки в $ФБ_i$, на случай, если следующий за ним функциональный блок еще не готов использовать этот результат. В рассмотренной схеме данные на вход конвейера могут подаваться с интервалом $T_{max}/3$(втрое чаще), и хотя задержка от момента поступления первой единицы данных в $Рг_{вх}$ до момента появления результата ее обработки на выходе $Рг_{вых}$ по-прежнему составляет T_{max}, последующие результаты появляются на выходе $Рг_{вых}$ уже с интервалом $T_{max}/3$.

На практике редко удается добиться того, чтобы задержки в каждом ФБ, были одинаковыми. Как следствие, производительность конвейера снижается, поскольку период поступления входных данных определяется максимальным временем их обработки в каждом функциональном блоке. Для устранения этого недостатка или, по крайней мере, частичной его компенсации каждый буферный регистр $Рг_i$, следует заменить буферной памятью $БП_i$ способной хранить множество данных и организованной по принципу FIFO - "первым вошел - первым вышел" (рис. 9.1, в). Обработав элемент данных, $ФБ_i$, заносит результат в $БП_i$ извлекает из $БП_{i-1}$ новый элемент данных и приступает к очередному циклу обработки, причем эта последовательность осуществляется каждым функциональным блоком независимо от других блоков. Обработка в каждом блоке может продолжаться до тех пор, пока не ликвидируется предыдущая очередь или пока не переполнится следующая очередь. Если емкость буферной памяти достаточно велика, различия во времени обработки не сказываются на производительности, тем не менее желательно, чтобы средняя длительность обработки во всех ФБ, была одинаковой.

В архитектуре вычислительных машин можно найти множество объектов, где конвейеризация обеспечивает ощутимый прирост производительности ВМ. Ранее уже рассматривались два таких объекта - операционные устройства и память, однако наиболее ощутимый эффект достигается при конвейеризации этапов машинного цикла.

По способу синхронизации работы ступеней конвейеры могут быть синхронными и асинхронными. Для традиционных ВМ характерны *синхронные конвейеры*. Связано это, прежде всего, с синхронным характером работы процессоров. Ступени конвейеров в процессоре обычно располагаются близко друг от друга, благодаря чему тракты распространения сигналов синхронизации получаются достаточно короткими и фактор «перекоса» сигналов становится не столь существенным. *Асинхронные конвейеры* оказываются полезными, если связь между ступенями не столь сильна, а длина сигнальных трактов между разными ступенями сильно разнится. Примером асинхронных конвейеров могут служить систолические массивы (систолическая обработка будет рассмотрена в последующих разделах).

Синхронные линейные конвейеры

Эффективность синхронного конвейера во многом зависит от правильного выбора длительности тактового периода $T_к$. Минимально допустимую $T_к$ можно определить как сумму наибольшего из времен обработки на отдельной ступени кон-

вейера T_{CMAX} и времени записи результатов обработки в буферные регистры между ступенями конвейера $T_{БР}$:

$$T_K \geq T_{CMAX} + T_{БР}.$$

Из-за вероятного «перекоса» в поступлении тактирующего сигнала в разные ступени конвейера предыдущую формулу следует дополнить еще одним элементом - максимальной величиной «перекоса» $T_{ПК}$:

$$T_K \geq T_{CMAX} + T_{БР} + T_{ПК}.$$

Каждая ступень может содержать множество логических трактов обработки. $T_к$ определяется наиболее длинными трактами во всех ступенях конвейера. При разработке конвейера необходимо учитывать, что для двух последовательных элементов, обрабатываемых одной и той же ступенью, обработка первого элемента может проходить по тракту максимальной длины, а второго элемента — по минимальному тракту. В итоге результат обработки второго элемента может появиться на выходе ступени прежде, чем в выходном регистре ступени будет запомнен предыдущий результат. Чтобы избежать подобной ситуации, сумма $T_{БР} + T_{ПК}$ должна быть меньше минимального времени обработки в ступени! T_{CMIN}, откуда

$$T_{БР} \leq T_{CMIN} - T_{ПК}.$$

Выбор длительности тактового периода для конвейера должен осуществляться с соблюдением соотношения

$$T_{CMAX} + T_{БР} + T_{ПК} \leq T_K \leq T_{CMAX} + T_{CMIN} - T_{ПК}.$$

Несмотря на очевидные преимущества выбора $T_к$ равным нижнему пределу, проектировщики ВМ обычно ориентируются на среднее значение между нижним и верхним пределами.

Метрики эффективности конвейеров

Чтобы охарактеризовать эффект, достигаемый за счет конвейеризации вычислений, обычно используют три метрики: ускорение, эффективность и производительность.

Под *ускорением* понимается отношение времени обработки без конвейера и при его наличии. Теоретически наилучшее время обработки входного потока из N значений T_{NK} на конвейере с K ступенями и тактовым периодом $T_к$ можно определить выражением

$$T_{NK} = (K + (N-1))T_K.$$

Формула отражает тот факт, что до появления на выходе конвейера результата обработки первого элемента должно пройти K тактов, а последующие результаты будут следовать в каждом такте.

В процессоре без конвейера общее время выполнения составляет NKT_K. Таким образом, ускорение вычислений S за счет конвейеризации вычислений можно описать формулой

$$S = \frac{NKT_K}{(K + (N-1))T_K} = \frac{NK}{K + (N-1)}.$$

При $N \to \infty$ ускорение стремится к величине, равной количеству ступеней в конвейере.

Еще одной метрикой, характеризующей конвейерный процессор, является *эффективность E* - доля ускорения, приходящаяся на одну ступень конвейера:

$$E = \frac{S}{K} = \frac{N}{K + (N - 1)}.$$

В качестве третьей метрики часто *выступает пропускная способность или производительность P* - эффективность, деленная на длительность тактового периода:

$$P = \frac{N}{T_K (K + (N - 1))}.$$

При $N \to \infty$ эффективность стремится к единице, а производительность — к частоте тактирования конвейера:

$$F \left(F = \frac{1}{T_K} \right).$$

Нелинейные конвейеры

Конвейер не всегда представляет собой линейную цепочку этапов. В ряде ситуаций оказывается выгодным, когда функциональные блоки соединены между собой не последовательно, а в соответствии с логикой обработки, при этом одни блоки в цепочке могут пропускаться, а другие — образовывать циклические структуры. Это позволяет с помощью одного и того же конвейера одновременно вычислять более одной функции, однако если эти функции конфликтуют между собой, то такой конвейер трудно загрузить полностью! Структура нелинейного конвейера, одновременно вычисляющего две функции *X и Y,* приведена на рис. 9.2. Там же показана последовательность, в которой функциями X и У востребуются те или иные функциональные блоки.

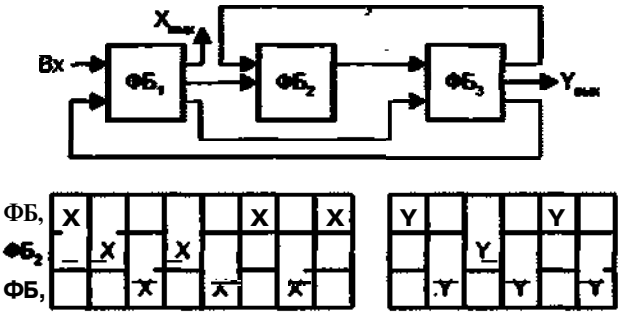


Рис. 9.2. Нелинейный конвейер

Чтобы определить, когда пора приступать к повторному вычислению той или иной функции, необходимо построить диаграмму однократной реализации этой функции и отследить по ней моменты, когда такой запуск не приведет к конфликту, связанному с одновременным обращением к одному и тому же функциональному блоку.

Так, в ходе реализации функции *X* запуск очередного ее вычисления возможен после 1,3 и 6 тактов. Запуск параллельного вычисления функции У возможен после

2 и 4 тактов. При запуске функции *У* очередной ее запуск позволен после тактов 1, 3 и 5, а параллельный запуск функции *X* допустим после 2 **и** 4 тактов.

Конвейер команд

Идея конвейера команд была предложена в 1956 году академиком С. АЛебедевым. Как известно, цикл команды представляет собой последовательность этапов. Возложив реализацию каждого из них на самостоятельное устройство и последовательно соединив такие устройства, мы получим классическую схему конвейера команд. Для иллюстрации воспользуемся примером, приведенным в [200]. Выделим в цикле команды шесть этапов:

1. **Выборка команды (ВК).** Чтение очередной команды из памяти и занесение ее в регистр команды.
2. **Декодирование команды (ДК).** Определение кода операции и способов адресации операндов.
3. **Вычисление адресов операндов (ВА).** Вычисление исполнительных адресов каждого из операндов в соответствии с указанным в команде способом их адресаций.
4. **Выборка операндов** (ВО). Извлечение операндов из памяти. Эта операция не нужна для операндов, находящихся в регистрах.
5. **Исполнение команды (ИК).** Исполнение указанной операции.
6. **Запись результата (ЗР),** Занесение результата в память.

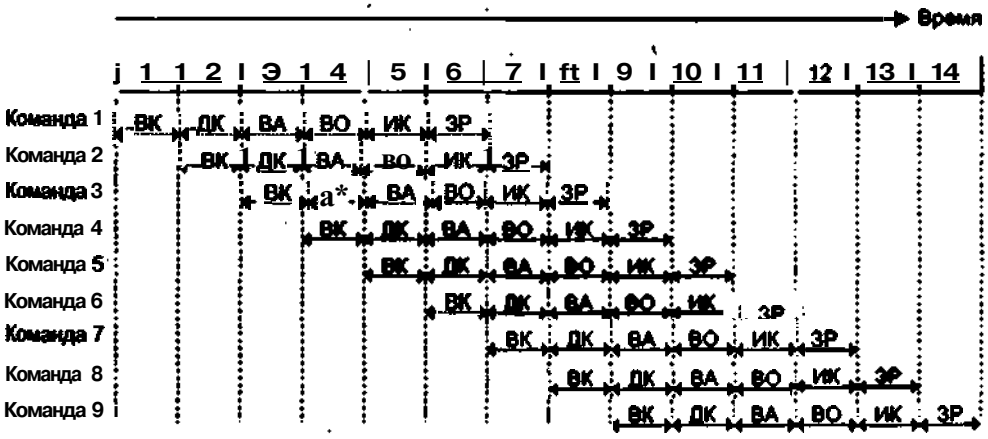


Рис. 9.3. Логика работы конвейера команд

На рис. 9.3 показан конвейер с шестью ступенями, соответствующими шести этапам цикла команды. В диаграмме предполагается, что каждая команда обязательно проходит все шесть ступеней, хотя этот случай не совсем типичен. Так, команда загрузки регистра не требует этапа ЗР. Кроме того, здесь принято, что все этапы могут выполняться одновременно. Без конвейеризации выполнение девяти команд заняло бы 9 x 6 = 54 единицы времени. Использование конвейера позволяет сократить время обработки до 14 единиц.

Конфликты в конвейере команд

Полученное в примере число 14 характеризует лишь потенциальную производительность конвейера команд. На практике в силу возникающих в конвейере конфликтных ситуаций достичь такой производительности не удается. Конфликтные ситуации в конвейере принято обозначать термином риск (hazard), а обусловлены они могут быть тремя причинами:

- попыткой нескольких команд одновременно обратиться к одному и тому же ресурсу ВМ (структурный риск);
- взаимосвязью команд по данным (риск по данным);
- неоднозначностью при выборке следующей команды в случае команд перехода (риск по управлению).

Структурный риск (конфликт по ресурсам) имеет место, когда несколько команд, находящихся на разных ступенях конвейера, пытаются одновременно использовать один и тот же ресурс, чаще всего — память. Так, в типовом цикле команды сразу три этапа (В К, 80 и ЗР) связаны с обращением к памяти. Диаграмма (см. рис. 9.3) показывает, что все три обращения могут производиться одновременно, однако на практике это не всегда возможно. Подобных конфликтов частично удается избежать за счет модульного Построения памяти и использования кэш-памяти - имеется вероятность того, что команды будут обращаться либо к разным модулям ОП, либо одна из них станет обращаться к основной памяти, а другая - к кэш-памяти. С этих позиций выгоднее разделять кэш-память команд и кэш-память данных. Конфликты из-за одновременного обращения к памяти могут и не возникать, поскольку для многих команд ступени выборки операнда и записи результата часто не требуются. В целом, влияние структурного риска на производительность конвейера по сравнению с другими видами рисков сравнительно невелико.

Риск по данным, в противоположность структурному риску — типичная и регуданным положим, что две команды в конвейере (i и j) предусматривают обращение к одной и той же переменной *x,* причем команда i предшествует команде). В общем случае между *i* и j ожидаемы три типа конфликтов по данным (рис. 9.4):

- *Чтение после записи»* (ЧПЗ): команда *j* читает *x* до того, как команда *i* успела записать новое значение *x,* то есть j ошибочно получит старое значение *x* вместо нового.
- *«Запись после чтения»* (ЗПЧ): команда; j записывает новое значение *x* до того, как команда *i* успела прочитать x:, то есть команда i ошибочно получит новое значение *x* вместо старого.
- *«Запись после записи»* (ЗПЗ): команда *j* записывает новое значение *x* прежде, чем команда i успела записать в качестве *x* свое значение, то есть *x* ошибочно содержит i-е значение *x* вместо *j*- го.

Возможен и четвертый случай, когда команда j читает x прежде команды *i.* Этот случай не вызывает никаких конфликтов, поскольку как i, так *j* получат верное значение *x.*

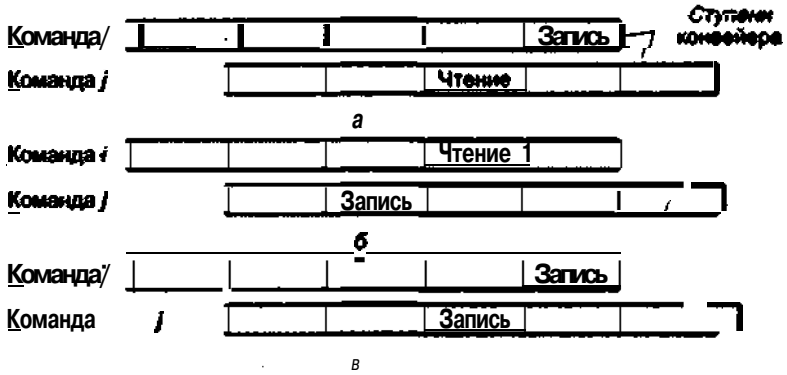


Рис- 9.4. Конфликты поданным: *а* - «Чтение после записи»'; *б* - -Запись после чтения-; в — «Запись после записи»

Наиболее частый вид конфликтов по данным - ЧПЗ, поскольку операция чтения в цикле команды (этап ВО) предшествует операции записи (этап ЗР). По той же причине конфликты типа ЗПЧ большой проблемы не представляют. Сложности появляются, только если структура конвейера допускает запись прежде чтения или если команды в конвейере обрабатываются в последовательности, отличной от предписанной программой. Такое возможно, если командам в конвейере разрешается «догонять» предшествующие им команды, приостановленные из-за какого-то конфликта. Конфликт типа ЗПЗ также не вызывает особых проблем в конвейерах, где команды следуют в порядке, определенном программой, и могут производить запись только на этапе ЗР. В худшем случае, когда одна команда догоняет другую из-за приостановки последней, имеет место конфликт по ресурсу — попытка одновременного доступа к одной и той же ячейке.

В борьбе с конфликтами по данным выделяют два аспекта: своевременное обнаружение потенциального конфликта и его устранение. Признаком возникновения конфликта по данным между двумя командами *i* и *j* служит невыполнение хотя бы одного из трех условий Бернстейна (Bernstein's Conditions):

- **для ЧПЗ:** $O_i \cap I(j) = 0$;
- **для ЗПЧ:** $I(i) \cap 0(f)$ - 0 ;
- **для ЗПЗ:** ft n $O(j) = 0$-

Где $O(k)$ - множество ячеек, изменяемых командой $к$; $1(1)$ - множество ячеек, читаемых командой l; **0 — пустое множество; ∩ — операция пересечения множеств.**

Критерий может быть распространен и на большее число команд: для трех команд подобных уравнений будет 9, для четырех команд - 18 (по три на каждую пару). Соблюдение соотношений является достаточным, но не необходимым условием, поскольку в ряде случаев коллизий может и не быть.

Для борьбы с конфликтами по данным применяются как программные, так и аппаратные методы.

Программные методы ориентированы на устранение самой возможности конфликтов еще на стадии компиляции программы. Оптимизирующий компилятор пытается создать такой объектный код, чтобы между командами, склонными к

конфликтам, находилась достаточное количество нейтральных в этом плане команд. Если такое не удается, то между конфликтующими командами компилятор вставляет необходимое количество команд типа «Нет операции»,

Фактическое разрешение конфликтов возлагается на аппаратные методы. Наиболее очевидным решением является остановка команды; на несколько тактов с тем, чтобы команда *i* успела завершиться или, по крайней мере, миновать ступень конвейера, вызвавшую конфликт. Соответственно задерживаются и команды, следующие в конвейере за j-й командой. Данную ситуацию называют *«пузырьком»* в конвейере. Иногда приостанавливают только команду *j,* не задерживая следующие за ней команды. Это более эффективный прием, но его реализация усложняет конвейер.

Понятно, что остановки конвейера снижают его эффективность и разработчики ВМ всячески стремятся сократить общее число остановок или хотя бы их длительность. Поскольку наиболее частые конфликты по данным - это ЧПЗ, основные усилия тратятся на противодействие именно этому типу конфликтов. Среди известных методов борьбы с ЧПЗ наибольшее распространение получил прием *ускоренного продвижения информации* (forwarding). Обычно между двумя соседними ступенями конвейера располагается буферный регистр, через который предшествующая ступень передает результат своей работы на последующую ступень, то есть передача информации возможна лишь между соседними ступенями конвейера. При ускоренном продвижении, когда для выполнения команды требуется операнд, уже вычисленный предыдущей командой, этот операнд может быть получен непосредственно из соответствующего буферного регистра, минуя все промежуточные ступени конвейера. С данной целью в конвейере предусматриваются дополнительные тракты пересылки информации (тракты опережения, тракты обхода), снабженные средствами мультиплексирования.

Наибольшие проблемы при создании эффективного конвейера обусловлены командами, изменяющими естественный порядок вычислений[1]. Простейший конвейер ориентирован на линейные программы. В нем ступень выборки извлекает команды из последовательных ячеек памяти, используя для этого счетчик команд (СК). Адрес очередной команды в линейной программе формируется автоматически, за счет прибавления к содержимому СК числа, равного длине текущей ко- В них обязательно присутствуют команды управления, изменяющие последова- возврат из процедуры и т. п. Доля подобных команд в программе оценивается как 10-20% (по некоторым источникам она существенно больше). Выполнение команд, изменяющих последовательность вычислений (в дальнейшем будем их называть командами перехода), может приводить к приостановке конвейера на несколько тактов, из-за чего производительность процессора снижается. Приостановки конвейера при выполнении команд перехода обусловлены двумя факторами.

[1] В фон-неймановской ВМ команды размещаются в ячейках памяти и извлекаются для выполнения в том же порядке, в каком они следуют в программе. Такую последовательность выполнения команд программы называют естественной.

Первый фактор характерен для любой команды перехода и связан с выборкой команды из точки перехода (по адресу, указанному в команде перехода). То, что текущая команда относится к командам перехода, становится ясным только после декодирования (после прохождения ступени декодирования), то есть спустя два такта от момента поступления Команды на конвейер. За это время на первые ступени конвейера уже поступят новые команды, извлеченные в предположении, что естественный порядок вычислений не будет нарушен. В случае перехода эти ступени нужно очистить и загрузить в конвейер команду, расположенную по адресу перехода, для чего нужен исполнительный адрес последней. Поскольку в командах перехода обычно указаны лишь способ адресации и адресный код, исполниступени конвейера. Таким образом, реализация перехода в конвейере требует определенных дополнительных операций, выполнение которых равносильно остановке конвейера как минимум на два такта.

Вторая причина нарушения ритмичности работы конвейера имеет отношение только к командам условного перехода Для пояснения сути проблемы воспользуемся ранее приведенной условной программой рис. 9.3), несколько изменив постановку задачи (рис. *9.5).*

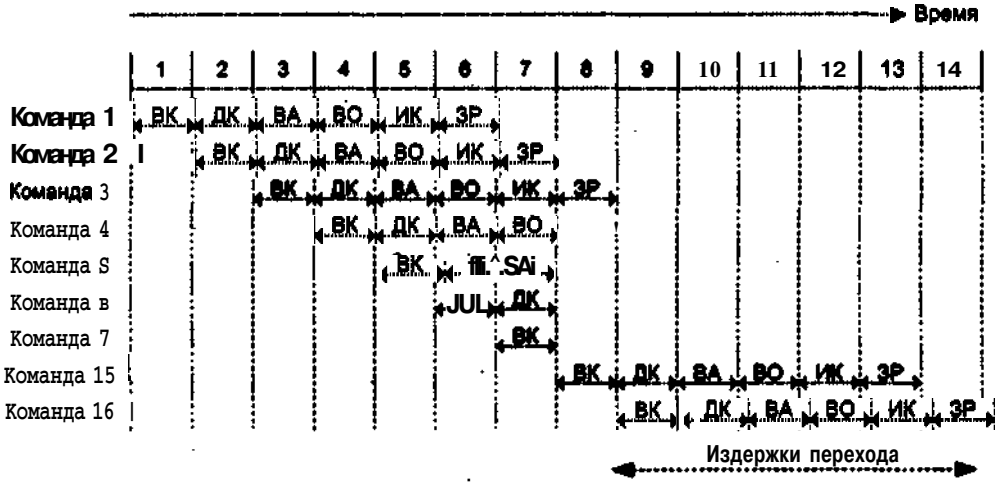


Рис. 9.5. Влияние условного перехода на работу конвейера команд

Пусть команда 3 - это условный переход к команде 15. До завершения команды 3 невозможно определить, какая из команд (4-я или 15-я) должна выполняться следующей, поэтому конвейер просто загружает следующую команду в последовательности (команду 4) и продолжает свою работу. В варианте, показанном на рис. 9.3, переход не произошел и получена максимально возможная производительность. На рис. 9.5 переход имеет место, о чем неизвестно до 7-го шага. В этой точке конвейер должен быть очищен от ненужных команд, выполнявшихся до данного момента. Лишь на шаге 8 на конвейер поступает нужная команда 15, из-за чего в течение тактов от 9 до 12 не будет завершена ни одна другая команда. Это и есть издержки из-за невозможности предвидения исхода команды условного

перехода. Как видно, они либо существенно больше, чем для прочих команд перехода (если переход происходит), либо отсутствуют вовсе (если переход не происходит).

Для сокращений задержек, обусловленных выборкой команды из точки перехода, применяются несколько подходов:

- вычисление исполнительного адреса перехода на ступени декодирования команды;
- использование буфера адресов перехода;
- использование кэш-памяти для хранения команд, расположенных в точке перехода;
- использование буфера цикла.

В результате декодирования команды выясняется не только ее принадлежность к командам перехода, но также способ адресации и адресный код точки перехода. Это позволяет сразу же приступить к вычислению исполнительного адреса перехода, не дожидаясь передачи команды на третью ступень конвейера, и тем самым сократить время остановки конвейера с двух тактов до одного. Для реализации этой идеи в состав ступени декодирования вводятся дополнительные сумматоры, с помощью которых и вычисляется исполнительный адрес точки перехода.

Буфер адресов перехода (BTB, Branch Target Buffer) представляет собой кэш-память небольшой емкости, в которой хранятся исполнительные адреса точек перехода нескольких последних команд, для которых переход имел место. В роли тегов выступают адреса соответствующих команд. Перед выборкой очередной команды ее адрес (содержимое счетчика команд) сравнивается с адресами команд, представленных в BTB. Для команды, найденной в буфере адресов перехода, исполнительный адрес точки перехода не вычисляется, а берется из BTB, благодаря чему выборка команды из точки перехода может быть начата на один такт раньше. Команда, ссылка на которую в BTB отсутствует, обрабатывается стандартным образом. Если это команда перехода, то полученный при ее выполнении исполнительный адрес точки перехода заносится в BTB, при условии, что команда завершилась переходом. При замещении информации в BTB обычно применяется алгоритм LRU.

Применение BTB дает наибольший эффект, когда отдельные команды перехода в программе выполняются многократно, что типично для циклов. Обычно BTB используется не самостоятельно, а в составе других, более сложных схем компенсации конфликтов по управлению.

Кэш-память команд, расположенных в точке перехода (BTIC, Branch Target Instruction Cache), - это усовершенствованный вариант BTB, где в кэш-память помимо исполнительного адреса команды в точке перехода записывается также и код этой команды. За счет увеличения емкости кэш-памяти BTIC позволяет при повторном выполнении команды перехода исключить не только этап вычисления исполнительного адреса точки перехода, но и этап выборки расположенной там команды. Преимущества данного подхода в наибольшей степени проявляются при многократном исполнении одних и тех же команд перехода, главным образом при реализации программных циклов.

Буфер цикла представляет собой маленькую быстродействующую память, входящую в состав первой ступени конвейера, где производится выборка команд. В буфере сохраняются коды « последних команд в той последовательности, в которой они выбирались. Когда имеет место переход, аппаратура сначала проверяет, нет ли нужной команды в буфере, и если это так, то команда извлекается из буфера. Стратегия наиболее эффективна при реализации циклов и итераций, чем и объясняется название буфера Если буфер достаточно велик, чтобы охватить все тело цикла, выборку команд из памяти достаточно выполнить только один раз в первой итерации, поскольку необходимые для последующих итераций команды уже находятся в буфере.

По принципу использования буфер цикла похож на BTIC, с той разницей, что в нем сохраняется последовательность выполнения команд, а сам буфер меньше по емкости и дешевле.

Среди ВМ, где реализован буфер цикла, можно упомянуть некоторые вычислительные машины фирмы CDC (Star 100,6600,7600) и суперЭВМ CRAY-1. Специализированная версия буфера цикла имеется и в микропроцессоре Motorola 68010, где он используется для особых циклов, включающих в себя команду «Уменьшение и переход по условию».

Методы решения проблемы условного перехода

Несмотря на важность аспекта вычисления исполнительного адреса точки перехода, основные усилия проектировщиков ВМ направлены на решение проблемы условных переходов, поскольку именно последние приводят к наибольшим издержкам в работе конвейера. Для устранения или частичного сокращения этих издержек были предложены различные способы, которые условно можно разделить на четыре группы:

- буферы предвыборки;
- множественные потоки;
- задержанный переход;
- предсказание перехода

Равномерность поступления команд на вход конвейера часто нарушается, например из-за занятости памяти или при выборке команд, состоящих из нескольких слов. Чтобы добиться ритмичности подачи команд на вход конвейера, между ступенью выборки команды и остальной частью конвейера часто размещают буферную память, организованную по принципу очереди (FIFO) и называемую *буфером предвыборки*. Буфер предвыборки можно рассматривать как несколько дополнительных ступеней конвейера. Подобное удлинение конвейера не сказывается на его производительности (при заданной тактовой частоте); оно лишь приводит к увеличению времени прохождения команды через конвейер. Типичные буферы предвыборки в известных процессорах рассчитаны на 2-8 команд.

Чтобы одновременно с обеспечением ритмичной работы конвейера решить и проблему условных переходов, в конвейер включают два таких буфера (рис. 9.6).

Каждая извлеченная из памяти и помещенная в основной буфер команда анализируется блоком перехода При обнаружении команды условного перехода (УП)

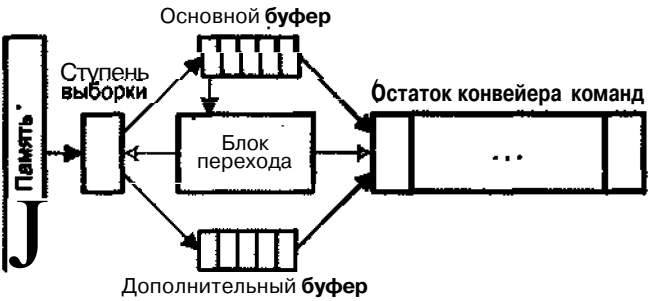


Рис. 9.6. Конвейер с буферами предвыборки команд

блок перехода вычисляет исполнительный адрес точки перехода и параллельно с продолжением последовательной выборки в основной буфер организует выборку в дополнительный буфер команд, начиная с точки УП. Далее блок перехода определяет исход команды УП, в зависимости от которого подключает к остатку конвейера нужный буфер, при этом содержимое другого буфера сбрасывается. Упрощенный вариант подобного подхода применен в IBM 360/91, где дополнительный буфер рассчитан на одну команду, то есть выигрыш достигается за счет исключения времени на выборку команды из точки перехода.

Помимо необходимости дублирования части оборудования, у метода имеется еще один недостаток. Так, если в потоке команд несколько команд УП следуют одна за другой или находятся достаточно близко, количество возможных ветвлений увеличивается и, соответственно, должно быть увеличено и число буферов предвыборки.

Другим решением проблемы переходов служит дублирование начальных ступеней конвейера и создание тем самым двух параллельных потоков команд (рис. 9.7).

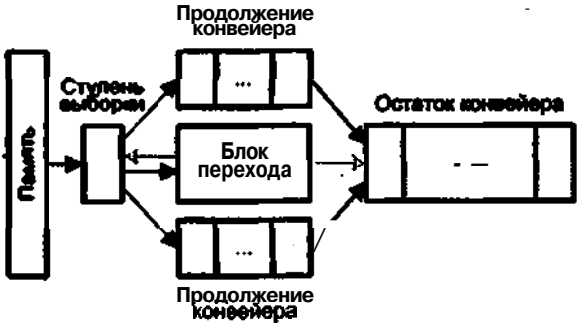


Рис. 9.7. Конвейер с множественными потоками

В одной из ветвей такого «раздвоенного» конвейера последовательность выборки и выполнения команд соответствует случаю, когда условие перехода не выполнилось, во второй ветви — случаю выполнения этого условия. Для ранее рассмотренного примера (см. рис. 9,5) в одном из потоков может обрабатываться последовательность команд 4-6, а в другом— 15-17. Оба потока сходятся в точке,

где итог проверки условия перехода становится очевидным. Дальнейшее продвижение по конвейеру продолжает только «правильный» поток. Основной недостаток метода состоит в том, что на конвейер или в поток может поступить новая команда У П до того, как будет принято окончательное решение по текущей команде перехода. Каждая такая команда требует дополнительного потока. Несмотря на это, стратегия позволяет улучшить производительность конвейера. Примерами ВМ, где используются два или более конвейерных потоков, служат I В М 370/168 и IBM 3033.

Стратегия *задержанного перехода* предполагает продолжение выполнения команд, следующих за командой УП, вне зависимости от ее исхода. Естественно, что это имеет смысл, лишь когда последующие команды являются «полезными», то есть такими, которые все равно должны быть когда-то выполнены, независимо от того, происходит переход или нет, и если команда перехода никак не влияет на результат их выполнения.

Для реализации этой идеи на этапе компиляции программы после каждой команды перехода вставляется команда «Нет операции». Затем на стадии оптимизации программы производятся попытки «перемешать» команды таким образом, чтобы по возможности большее количество команд «Нет операции»- заменить «полезными» командами программы. Разумеется, замещать команду «Нет операции» можно лишь на такую, которая не влияет на условие выполняемого перехода, иначе полученная последовательность команд не будет функционально эквивалентной исходной. В оптимизированной программе удается заменить более 20% команд «Нет операции» [120].

Предсказание переходов

Предсказание переходов на сегодняшний день рассматривается как один из наиболее эффективных способов борьбы с конфликтами по управлению. Идея заключается в том, что еще до момента выполнения команды условного перехода или сразу же после ее поступления на конвейер делается предположение о наиболее вероятном исходе такой команды (переход произойдет или не произойдет). Последующие команды подаются на конвейер в соответствии с предсказанием. Для иллюстрации вернемся к примеру (см. рис. 9.5), где команда 3 является командой УП. Пусть для команды 3 предсказано, что переход не произойдет. Тогда вслед за командой 3 на конвейер будут поданы команды 4-6 и т. д. Если предсказан переход, то после команды 3 на конвейер подаются команды 15-17,... При ошибочном предсказании конвейер необходимо вернуть к состоянию, с которого началась выборка «ненужных» команд (очистить начальные ступени конвейера), и приступить к загрузке, начиная с «правильной» точки, что по эффекту эквивалентно приостановке конвейера Цена ошибки может оказаться достаточно высокой, но при правильных предсказаниях крупен и выигрыш — конвейер функционирует ритмично без остановок и задержек, причем выигрыш тем больше, чем выше точность предсказания. Термин *«точность предсказаниям »в* дальнейшем будем трактовать как процентное отношение числа правильных предсказаний к их общему количеству. В работе [70] дается следующая оценка: чтобы снижение производительности конвейера из-за его остановок по причине конфликтов по управлению не превысило 10%, точность предсказания переходов должна быть выше 97,7%.

К настоящему моменту известно более двух десятков различных способов реализации идеи предсказания переходов [165,230-232], отличающихся друг от друга исходной информацией, на основании которой делается прогноз, сложностью реализации и, главное, точностью предсказания. При классификации схем предсказания переходов обычно выделяют два подхода: статический и динамический, в зависимости от того, когда и на базе какой информации делается предсказание.

Эффективность большинства из приводимых в учебнике методов предсказания переходов иллюстрируется результатами исследований, опубликованными в [68,95,107,197,207,228]. Все эксперименты проводились по примерно одинаковой методике: численные показатели получены путем имитации методов предсказания переходов при выполнении наборов стандартных тестовых программ. Главное различие заключалось в выборе тестовых программ, что и нашло отражение в существенном расхождении полученных оценок.

Так, в работе Смита [197] использовались шесть тестовых программ, написанных на языке Фортран:

- ADVAN: решение системы из трех дифференциальных уравнений в частных производных;
- GIBSON: искусственная программа компиляции набора команд, примерно удовлетворяющего так называемой смеси Гибсона № 5;
- SCI2: обращение матрицы;
- SINCOS: преобразование массива координат из полярной системы отсчета в прямоугольную;
- SORTST: сортировка массива из 10 000 целых чисел;
- TBLINK: работа со связанным списком.

В прочих исследованиях участвовали программы, входящие в различные версии тестовых пакетов SPEC, в частности пакетов SPEC_92, SPEC_95 и CPU2000.

Последующий материал раздела посвящен рассмотрению различных механизмов предсказания переходов.

Статическое предсказание переходов

Статическое предсказание переходов осуществляется на основе некоторой априорной информации о подлежащей выполнению программе. Предсказание делается на этапе компиляции программы и в процессе вычислений уже не меняется. Главное различие между известными механизмами статического прогнозирования заключается в виде информации, используемой для предсказания, и ее трактовке. Исходная информация может быть получена двумя путями: на основе анализа кода программы или в результате ее профилирования (термин «профилирование» поясняется ниже).

Известные стратегии статического предсказания для команд УП можно классифицировать следующим образом:

- переход происходит всегда (ПВ);
- переход не происходит никогда (ПН);
- предсказание определяется по результатам профилирования;

- предсказание определяется кодом операции команды перехода;
- предсказание зависит от направления перехода;
Я при первом выполнении команды переход имеет место всегда,

В первом иэ перечисленных вариантов предполагается, что *каждая команда условного перекода в программе обязательно завершится переходом, и, с учетом* такого предсказания, дальнейшая выборка команд производится, начиная с адреса перехода В основе второй стратегии лежит прямо противоположный подход: *ни одна из команд условного перехода в программе никогда не завершается переходом,* поэтому выборка команд продолжается в естественной последовательности.

Интуитивное представление, что обе стратегии должны приводить к верному предсказанию в среднем в 50% случаев, на практике не подтверждается. Так, по результатам тестирования (рис. 9.8), приведенным в [197], предсказание об обязательном переходе оказалось правильным в среднем для 76% команд УП.

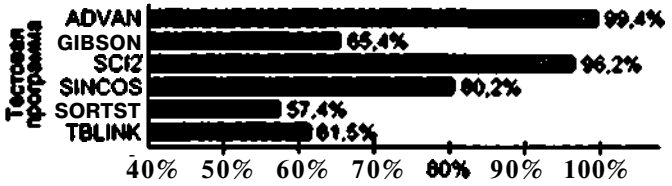


Рис. в.8. Точность предсказания переходов при стратегии «переход происходит всегда-

Аналогичный показатель, полученный на ином наборе тестовых программ [150], составил 68%. В работе [228] приводятся результаты проверки стратегии ПВ на тестах CPU2000 отдельно для программ с интенсивными целочисленными вычислениями и программ с преимущественной обработкой чисел в форме с плавающей запятой. Для первых средняя точность предсказаний составила 55t2%, а для вторых — 59,9%. В других источниках встречаются и иные численные оценки точности прогноза при стратегии ПВ, в частности 60% и 71,9%. В то же время в работе [233]. отмечается, что для ряда программ, связанных с целочисленной обработкой, процент правильных предсказаний может опускаться ниже 50%.

Цифры свидетельствуют, что успешность стратегии ПВ существенно зависит от характера программы и методов программирования, что, естественно, можно рассматривать как недостаток. Тем не менее стратегия все же используется в ряде ВМ, в частности MIPS-X, SuperSPARC, микропроцессорах i486 фирмы Intel. Связано это, скорее всего, с простотой реализации и с тем, что для определенного класса программ стратегию можно считать достаточно эффективной.

Схожая ситуация характерна и для стратегии ПН, где предполагается, что ни одна из команд УП в программе никогда не завершается переходом. Несмотря на схожесть с ПВ, процент правильных исходов здесь обычно ниже, особенно в программах с большим количеством циклов.

Стратегия ПН реализована в конвейерах микропроцессоров M68020 и MC88000, вычислительной машине VAX 11/780. Сравнительная оценка стратегий ПВ и ПН, полученная при выполнении тестов CPU2000 для целочисленных и вещественных вычислений [228], показана на рис. 9.9.

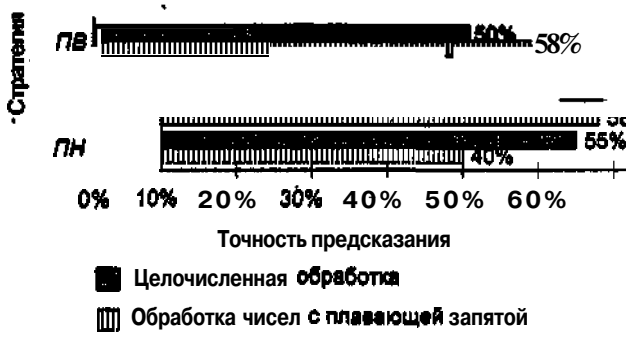


Рис. 9.9. Сравнение статических стратегий ПВ и ПН

Из диаграммы видно, что ни одна из двух стратегий не обладает явным преимуществом над другой. Тем не менее анализ большого количества программ [154] показывает, что условные переходы имеют место более чем в 50% случаев, поэтому если стоимость реализации двух рассмотренных вариантов одинакова, то предпочтение следует отдать стратегии ПВ.

В третьем из перечисленных способов статического предсказания назначение командам УП наиболее вероятного исхода производится *по результатам профилирования подлежащих выполнению программ. Под профилированием* подразумевается выполнение программы при некотором эталонном наборе исходных данных, сопровождающееся сбором информации об исходах каждой команды условного перехода. Тем командам, которые чаще завершались переходом, назначается стратегия ПВ, а всем остальным - ПН. Выбор фиксируется в специальном бите кода операции. Некоторые компиляторы самостоятельно проводят профилирование программы и по его результатам устанавливают этот бит в формируемом объектном коде. При выполнении программы поведение конвейера команд определяется после выборки команды по состоянию упомянутого бита в коде операции. Основной недостаток этого образа действий связан с тем, что изменение набора исходных данных для профилирования может существенно менять поведение одних и тех же команд условного перехода.

Средняя вероятность правильного предсказания, полученная на программах тестового набора SPEC_92, составила 75%. Стратегия используется в процессорах MIPS 4x00 и PowerPC 603.

При предсказании на основе кода операции команды перехода для одних кодов предполагается, что переход произойдет, для других - его не случится. Например, для команды перехода по переполнению логично предположить, что перехода не будет. На практике назначение разным видам команд УП наиболее вероятного исхода чаще производится по результатам профилирования программ. В исследованиях Е. Смита [197] стратегия ПВ была назначена командам перехода по условиям: «меньше нуля», «равно», «больше или равно», а ПН — всем прочим командам условного перехода. Результаты показаны на рис. 9.10.

Эффективность предсказания зависит от характера программы. Этим, в частности, объясняется различие в выводах, полученных на основе разных исследований. Так, согласно [154] стратегия приводит к успеху в среднем более чем в 75% случаев, а исследования Е. Смита дают цифру 86,7%,

Достаточно логичным представляется предсказание *исходя из направления перехода*. Если указанный в команде адрес перехода меньше содержимого счетчика

Рис. 9.10. Точность предсказания переходов при стратегии «переход зависит от кода операции»

команд, говорят о переходе «назад»-, и для такой команды назначается стратегия ПВ. Переход к адресу, превышающему адрес команды перехода, считается переходом «вперед», и такой команде назначается стратегия ПН. В основе рассматриваемого подхода лежит статистика по множеству программ, согласно которой большинство команд У П в программах используются для организации циклов, причем, как правило, переходы происходят к началу цикла (переходы «назад»). Согласно [120], для команд, выполняющих переход -«назад», фактический переход имеет место в 85% случаев. Для переходов «вперед» эта цифра составляет 65%. Таким образом, для команд условного перехода «назад»- логично принять, что переход происходит всегда. Рассматриваемую стратегию иногда называют «Переход назад происходит всегда». Результаты ее оценки [197] приведены на рис. 9.11.

Рис. 9.11. Точность предсказания переходов при стратегии «переход зависит от направления перехода»

Здесь при тестировании на пакете SPEC_89 средняя точность предсказания составила 65%. Среди ВМ, где описанная стратегия является основной, можно упомянуть MicroSPARC-2 и PA-7xOO. Чаще, однако, стратегия используется в качестве вторичного механизма в схемах динамического прогнозирования переходов.

В последней из рассматриваемых статических стратегий при *первом выполнешшлюбой команды условного перехода делается предсказание, что переход обязательно будет*. Предсказания на последующее выполнение команды зависят от правильности начального предсказания. Стратегию можно считать статической только частично, поскольку она однозначно определяет исход команды лишь при первом ее выполнении. Точность прогноза в соответствии с данной стратегией выше, чем у всех предшествующих, что подтверждают результаты, приведенные на рис. 9.12 [197].

К сожалению, при больших объемах программ вариант практически нереализуем из-за того, что нужно отслеживать слишком много команд условного перехода.

Рис. 9.12. Точность предсказания переходов при стратегии «при первом выполнении переход обязательно происходит»

Динамическое предсказание переходов

В *динамических стратегиях* решение о наиболее вероятном исходе команды УП принимается в ходе вычислений, исходя из информации о предшествующих переходах (истории переходов), собираемой в процессе выполнения программы. В целом, динамические стратегии по сравнению со статическими обеспечивают более высокую точность предсказания. Прежде чем приступить к рассмотрению конкретных динамических механизмов предсказания переходов, остановимся на некоторых положениях, общих для всех динамических подходов.

Идея динамического предсказания переходов предполагает накопление информации об исходе предшествующих команд УП. История переходов фиксируется в форме таблицы, каждый элемент которой состоит из *m* битов. Нужный элемент таблицы указывается с помощью k-разрядной двоичной комбинации - шаблона (pattern). Этим объясняется общепринятое название таблицы предыстории *переходов - таблица истории для шаблонов (PHT,* Pattern History Table).

При описании динамических схем предсказания переходов их часто расценивают как один из видов автоматов Мура, при этом содержимое элементов PHT трактуется как информация, отображающая текущее состояние автомата. В работе [230] рассмотрены различные варианты подобных автоматов, однако реально осмысленно говорить лишь о трех вариантах, которые условно обозначим Al, A2 и A3.

Автомат Al имеет только два состояния, поэтому каждый элемент PHT состоит из одного бита (m= 1), значение которого отражает исход последнего выполнения команды условного перехода. Диаграмма состояний автомата Al приведена па рис. 9.13.

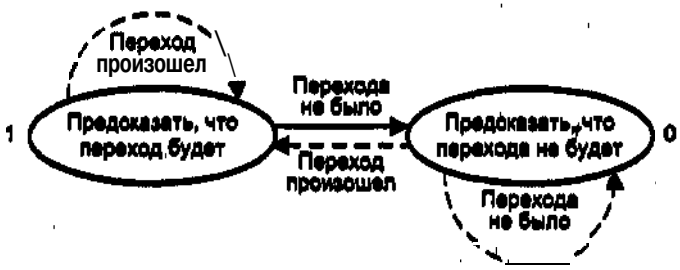


Рис. 0.13. Диаграмма состояний автомата A1

Если команда завершилась переходом, то в соответствующий элемент РНТ заносится единица, иначе — ноль. Очередное предсказание совпадает с итогом предыдущего выполнения команды. После обработки очередной команды содержимое элемента корректируется.

Два других автомата предполагают большее число состояний, поэтому в них используются РНТ с многоразрядными элементами. Чаще всего ограничиваются двумя разрядами (m =2) и, соответственно, автоматами с четырьмя состояниями.

В двухразрядном автомате А2 элементы РНТ отражают исходы двух последних выполненных команд условного перехода и заполняются по схеме регистра сдвига. После обработки очередной команды УП содержимое выделенного этой команде элемента РНТ сдвигается влево на один разряд, а в освободившуюся позицию заносится единица (если переход был) или ноль (если перехода не было). Если в элементе РНТ присутствует хотя бы одна единица, то при очередном выполнении команды делается предсказание, что переход будет. При нулевом значении элемента РНТ считается, что перехода не будет. Диаграмма состояний для такого автомата показана на рис. 9.14.

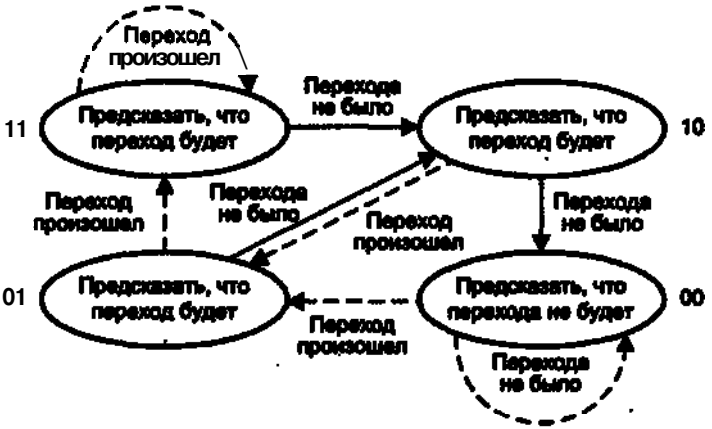


Рис. 9.14. Диаграмма состояний двухразрядного автомата А2

Двухразрядная схема автомата А2 используется относительно редко. Большую известность получила трехразрядная модель. Она, в частности, реализована в процессоре HP PA 8000.

Элементы РНТ в автомате А3 можно рассматривать как реверсивные счетчики, работающие в режиме с насыщением. При поступлении на конвейер команды условного перехода происходит обращение к соответствующему счетчику РНТ, и в зависимости от текущего состояния счетчика делается прогноз, определяющий дальнейший порядок извлечения команд программы. После прохождения ступени исполнения, когда фактический исход команды становится ясным, содержимое счетчика увеличивается на единицу (если команда завершилась переходом) или уменьшается на единицу (если перехода не было). Счетчики работают в режиме насыщения. Это означает, что добавление единиц сверх максимального числа в счетчике, а также вычитание единиц при нулевом содержимом счетчика уже не

производятся. Основанием для предсказания служит состояние старшего разряда счетчика. Если он содержит единицу, то принимается, что переход произойдет, в противном случае предполагается, что перехода не будет.

Интуитивное представление о том, что с увеличением глубины предыстории (увеличением *m)* точность предсказания должна возрастать, на практике не подтверждается. На рис. 9.15 показаны результаты одного из многочисленных исследований, свидетельствующие, что при $m > 3$ точность предсказания начинает снижаться.

Рис. 9.15. Зависимость точности предсказания от разрядности элементов PHT

График показывает также, что различие в точности предсказания при $m = 3$ и $m = 2$ незначительно, что удостоверяют также результаты, приведенные в [197] (рис. 9.16).

Рис. 9.16. Точность предсказания при использовании в PHT двухразрядных и трехразрядных счетчиков

Как следствие, в большинстве известных процессоров используются двухразрядные счетчики (m= 2). Логика предсказания переходов применительно к двухразрядным счетчикам известна как *алгоритм Смита* [193]. Алгоритм предполагает четыре состояния счетчика:

- 00 - перехода не будет (сильное предсказание);
- 01 - перехода не будет (слабое предсказание);
- 10 - переход будет (слабое предсказание);
- 11 - переход будет (сильное предсказание).

Рис. 9.17. Диаграмма состояний двухразрядного автомата А3

Логику предсказания можно описать диаграммой состояний двухразрядного автомата А3 (рис. 9.17).

Поскольку вариант РНТ со счетчиками получил наиболее широкое распространение, в дальнейшем будем ссылаться именно на него.

После определения способов учета истории переходов и логики предсказания необходимо остановиться на особенностях использования таблицы, в частности на том, какая информация выступает в качестве шаблона для доступа к РНТ и какого рода история фиксируется в элементах таблицы. Именно различия в способах использования РНТ определяют ту или иную стратегию предсказания.

В качестве шаблонов для доступа к РНТ могут быть взяты:

- адрес команды условного перехода;
- регистр глобальной истории;
- регистр локальной истории;
- комбинация предшествующих вариантов.

Схема, где для доступа к РНТ выбран адрес команды условного перехода (содержимое счетчика команд), позволяет учитывать поведение каждой конкретной команды УП. При многократном выполнении большинства команд условного перехода наблюдается повторяемость исхода: переход либо, как правило, происходит, либо, как правило, не происходит (имеет место *бимодальное распределение исходов)*. Индексация РНТ с помощью адреса команды УП дает возможность отделить первые от вторых и, соответственно, повысить точность предсказания. Каждой команде условного перехода в РНТ соответствует свой счетчик. Когда команда завершается переходом, содержимое счетчика увеличивается на единицу, а в противном случае — уменьшается на единицу (естественно, с соблюдением логики счета с насыщением). В качестве шаблона для поиска в РНТ служат младшие *k* разрядов содержимого счетчика команд (рис. 9.18).

При k-разрядном индексе таблица может содержать 2^k элементов.

Схема обеспечивает достаточно высокий процент правильных предсказаний для тех команд У П, которые в ходе вычислений выполняются многократно, например

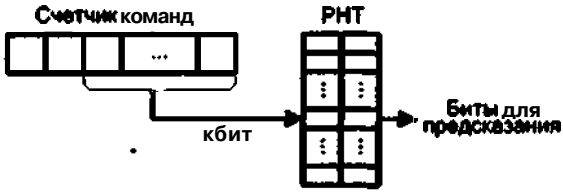


Рис. 9.18. Индексирование РНТ с помощью адреса команды перехода

предназначены для управления циклом. В то же время в любой программе имеется достаточно много команд перехода, выполняемых лишь однократно или малое число раз. Как показали исследования, исход для таких команд в значительной мере зависит от поведения предшествующих им команд УП, связь которых с рассматриваемой командой не столь очевидна. Иными словами, между исходами команд условного перехода в программе существует известная взаимосвязь, учет которой дает возможность повысить долю правильных предсказаний. Эта идея реализуется схемой с регистром глобальной истории.

Регистр глобальной истории (GHR, Global History Register) представляет собой l-разрядный сдвиговый регистр (рис. 9.19). После выполнения очередной команды условного перехода содержимое регистра сдвигается на один разряд, а в освободившуюся позицию заносится единица (если исходом команды был переход) или ноль (если перехода не было). Следовательно, кодовая комбинация в GHR отражает историю выполнения последних l команд условного перехода. Под индексирование массива предикторов (элементов механизма предсказания) выделяются *k* младших разряда GHR (чаще всего l - *k).* Каждой k-разрядной комбинации исходов последовательно выполнявшихся команд У П в массиве дескрипторов соответствует свой счетчик. Таким образом, счетчик РНТ определяется тем, какая комбинация исходов имела место в *k* предшествовавших командах перехода. Содержимое счетчика используется для предсказания исхода текущей команды перехода и впоследствии модифицируется по результату фактического выполнения команды.

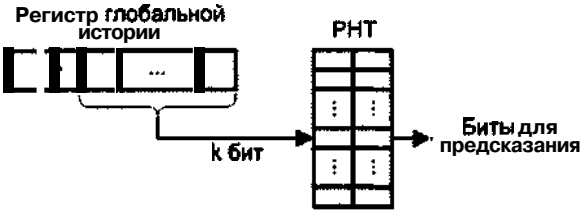


Рис. 9.19. Индексирование РНТ с помощью регистра глобальной истории

Регистр локальной истории(LHR, Local History Register) по логике работы аналогичен регистру глобальной истории, но предназначен для фиксации последовательных исходов одной и той же команды УП. В схемах предсказания с LHR присутствует так называемая *таблица локальной истории,* представляющая собой массив регистров локальной истории (рис. 9.20).

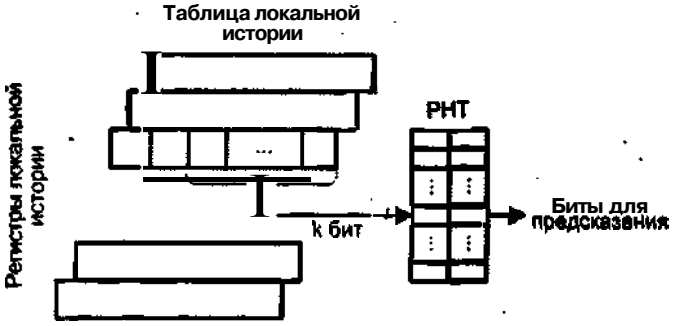


Рис. 9.20. Индексирование PHT с помощью регистров локальной истории

Как и в схеме с адресом команды перехода, каждый счетчик в PHT фиксирует историю исхода только одной команды УП, но базируясь на более детальных знаниях, отражающих к тому же и последовательность исходов.

Ранее уже отмечалось, что действие команды условного перехода зависит не только от результатов предшествующих выполнений данной команды, но и от исхода других команд перехода. Учет обоих факторов позволяет повысить точность предсказаний. С этой целью в ряде динамических методов предсказания шаблон для доступа к PHT формируется путем объединения адреса команды перехода и содержимого GHR (либо LHR), при этом используется одна из двух операций: конкатенация (сцепление) и сложение по модулю 2 («исключающее ИЛИ»),

При конкатенации k-разрядный шаблон для обращения к PHT образуется посредством взятия *q* младших битов из одного источника, к которым пристыковываются *k-q* младших разрядов, взятых из второго источника (рис, 9.21).

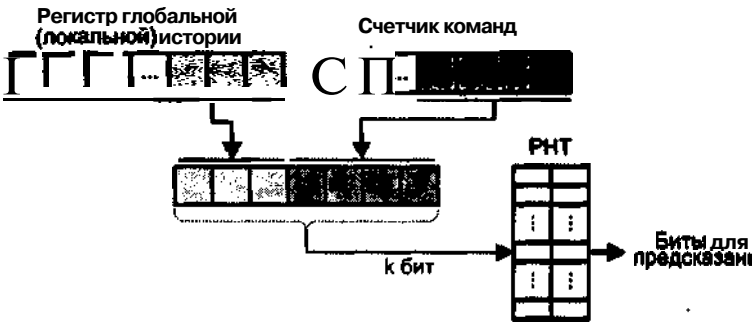


Рис. 9.21. Формирование шаблона для доступа к PHT путем конкатенации

Эффективность предсказания на основе подобного шаблона зависит от соотношения количества разрядов *(q* и *k-q),* выбранных от каждого из двух источников. Здесь многое определяется и характером программы. В качестве иллюстрации на рис. 9.22 приведена зависимость точности предсказания от соотношения числа битов, взятых от счетчика команд (СК) и регистра глобальной истории (GHR). Данные получены на тестовой программе xlisp (интерпретатор языка LISP) при объеме PHT, равном 1024 входам.

Рис. 9.22. Зависимость точности предсказания от соотношения разрядов в шаблоне при конкатенации [197]

Вариант со сложением по модулю 2 предполагает побитовое применение операции «Исключающее ИЛИ» к обоим источникам (рис. 9,23). В качестве шаблона используются *к* младших разрядов результата. Сформированный шаблон содержит больше полезной информации для предсказания, чем каждый из источников по отдельности.

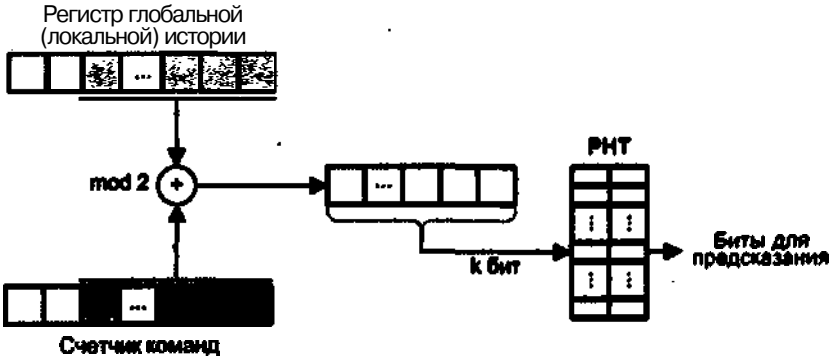


Рис. 9.23. Формирование шаблона для доступа к PHT путем сложения по модулю 2

Для сравнительной оценки рассмотренных вариантов доступа к PHT обратимся к результатам экспериментов, приведенным в [64]. Исследовались четыре тестовых программы:

- compress- сжатиефайлов(10млнУП);
- eqntott - преобразование логических функций, заданных таблицей истинности(178млнУП);
- espresso — минимизация логических функций (73 млн УП);
- xlisp - интерпретатор LISP (772 млн УП).

Результаты, усредненные по всем четырем программам для различных по объему таблиц PHT, показаны на рис. 9.24.

Первый вывод, вытекающий из представленных данных, очевиден — с увеличением размера PHT точность предсказания возрастает. Среди четырех рассмотренных схем наилучшую точность обеспечивает схема со сложением по модулю два. Схема со счетчиком команд относится к наименее эффективным. Объясняется это тем, что каждый отдельный счетчик PHT в схеме со счетчикок команд задействуется значительно реже, а некоторые счетчики вообще остаются без внима-

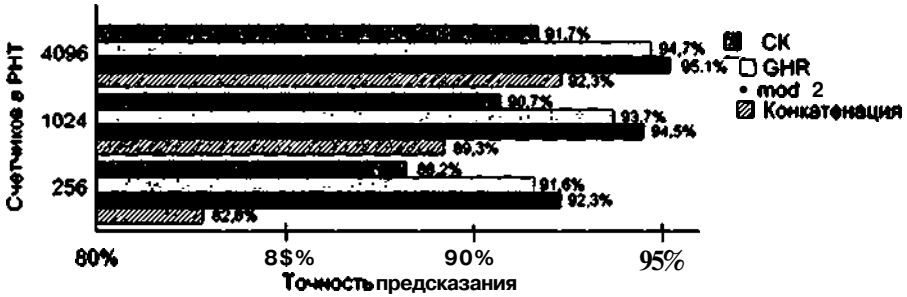


Рис. 9.24. Зависимость точности предсказания от способа доступа к PHT и размера таблицы

ния. Результаты для схемы с конкатенацией получены для случая, когда в шаблоне используется одинаковое число разрядов из CK и GHR. При малых объемах PHT вариант с конкатенацией дает наихудшие результаты, но при больших PHT эта схема превосходит модель со счетчиком команд.

В реальных схемах предсказания переходов размер таблицы PHT ограничен. Типичное количество элементов PHT (элементарных счетчиков) в разных процессорах варьируется от 256 до 4096. Для выбора определенного входа в PHT (нужного счетчика) применяется 6-разрядный шаблон, где *k* определяется размером массива. Для упомянутых выше размеров PHT значение *k* лежит в диапазоне от 8 до 12. Если обращение к PHT определяется счетчиком команд, разрядность которого обычно больше, чем *k,* в качестве шаблона выступают *k* младших битов CK. Как следствие, две команды условного перехода, адреса которых в младших *k* разрядах совпадают, будут обращаться к одному и тому же элементу PHT, и история выполнения одной команды будет накладываться на историю выполнения другой, что, естественно, будет влиять на точность предсказания. Ситуация известна *ка* к *эффект наложения* (aliasing). Та же проблема существует и при доступе к PHT на основании содержимого регистра глобальной истории или регистра локальной истории. В зависимости от типа программы и других факторов наложение может приводить к повышению точности предсказания, ее ухудшению либо вообще не сказываться на точности предсказания. Соответственно, эффекты наложения класси*фицируют как конструктивный, деструктивный и нейтральный.*

Рассмотрим, насколько часто предсказания производятся на основании тех счетчиков PHT, при обращении к которым имел место эффект наложения (рис. 9.25) [64].

Рис. 9.25. Интенсивность наложения при различных способах доступа к PHT

В больших PHT частота перекрытия снижается по причине большего числа счетчиков. Схема со счетчиком команд в наименьшей степени подвержена эффекту наложения. С другой стороны, в схемах, где шаблон для доступа к PHT формируется на основе содержимого регистра глобальной истории, причем используется операция конкатенации, эффект наложения сказывается в значительной мере и обычно отрицательно влияет на точность предсказания переходов.

При классификации динамических стратегий обычно выделяют следующие их виды:

- одноуровневые или бимодальные;
- двухуровневые или коррелированные;
- гибридные;
- асимметричные.

Одноуровневые схемы предсказания переходов. В многочисленных исследованиях, проводившихся на самых разнообразных программах, была отмечена интересная закономерность. В поведении многих команд условного перехода явно прослеживается тенденция повторяемости исхода: одни команды программы, как правило, завершаются переходом, в то время как другие — остаются без оного, то есть имеет место бимодальное распределение исходов. Идея одноуровневых схем предсказания, известных также под вторым названием — *бимодальные схемы* [165], сводится к отделению команд, имеющих склонность завершаться переходом, от команд, при выполнении которых переход обычно не происходит. Такая дифференциация позволяет для каждой команды выбрать наиболее подходящее предсказание. Для реализации идеи в составе схемы предсказания достаточно иметь лишь одну таблицу, каждый элемент которой отображает историю исходов одной команды УП. Для обращения к элементу, ассоциированному с определенной командой УП, используется адрес этой команды (или его младшие биты). Таким образом, одноуровневые схемы предсказания переходов можно определить как схемы, содержащие один уровень таблиц истории переходов (обычно единственную таблицу), адресуемых с помощью адреса команды условного перехода.

В первом из возможных вариантов одноуровневых схем строится сравнительно небольшая таблица, куда заносятся адреса *n* последних команд У П, при выполнении которых переход не случился. В сущности, это ранее упоминавшийся *буфер адресов перехода* (BTB), но с противоположным правилом занесения информации (фиксируются адреса команд, завершившихся без перехода). Таблица обычно реализуется на базе ассоциативной кэш-памяти. При поступлении на конвейер очередной команды УП ее адрес сравнивается с адресами, хранящимися в таблице. При совпадении делается предположение о том, что перехода не будет, в противном случае предсказывается переход. С этих позиций для каждой новой команды УП по умолчанию принимается, что переход произойдет. После того как фактический исход становится очевидным, содержимое таблицы корректируется. Если команда завершилась переходом, соответствующая запись из таблицы удаляется. Если же перехода не было, то адрес команды заносится в таблицу при условии, что до этого он там отсутствовал. При заполнении таблицы опираются на алгоритмы LRU или FIFO. По точности предсказания стратегия превосходит большинство стратегий статического предсказания.

Вторая схема ориентирована на то, что команды программы извлекаются из кэш-памяти команд. Каждая ячейка кэш-памяти содержит дополнительный разряд, который используется только применительно к командам условного перехода. Состояние разряда отражает исход предыдущего выполнения команды (1 — переход был, 0 - перехода не было). Новое предсказание совпадает с результатом предшествующего выполнения данной команды. При занесении такой команды в кэш-память рассматриваемый разряд устанавливается в единицу. Это напоминает стратегию «при первом выполнении переход обязательно происходит» с той лишь разницей, что первое предсказание, хотя и носит статический характер, происходит в ходе заполнения кэш-памяти, то есть динамически. После выполнения команды состояние дополнительного бита корректируется: если переход имел место, в него заносится единица, а в противном случае — ноль. Эффективность стратегии характеризуют данные, полученные в [197] (рис. 9.26).

Рис. 9.26, Точность предсказания схемы с дополнительным битом в кэш-памяти команд

Главный ее недостаток заключается в дополнительных затратах времени на обновление состояния контрольного разряда в кэш-памяти команд.

В третьей схеме (рис. 9.27) РНТ состоит из одноразрядных элементов и носит название *таблицы истории переходов* (BHT, Branch History Table).

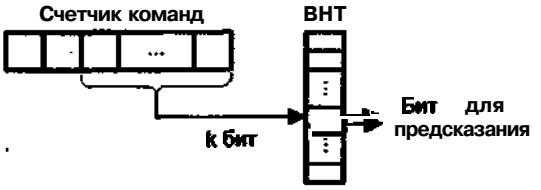


Рис. 9.27. Однобитовая бимодальная схема предсказания

Состояние элемента BHT определяет, произошел ли переход в ходе последнего выполнения команды условного перехода (1) или нет (0). Каждой команде УП в BHT соответствует свой элемент, для обращения к которому используются *k* младших разрядов адреса команды. Предсказание совпадает с исходом предыдущего выполнения команды. Если команда условного перехода участвует в организации цикла, то стратегия всегда приводит к неправильному предсказанию перехода в первой и последней итерациях цикла.

Схема была реализована в процессорах Alpha 21064 и AMD K5. Согласно результатам большинства исследовании, средняя точность успешного прогноза с помощью однобитовой бимодальной схемы не превышает 78%. В то же время в работе [197] получено значение 90,4%.

Точность предсказания перехода существенно повышается с увеличением разрядности элементов BHT. В четвертой из рассматриваемых одноуровневых схем каждый элемент BHT состоит из двух битов, выполняющих функцию двухразрядного счетчика, работающего в режиме с насыщением. Иными словами, реализуется алгоритм Смита. Каждый-счетчик отображает историю выполнения одной команды УП, то есть адресуется младшими разрядами счетчика команд (рис. 9.28).

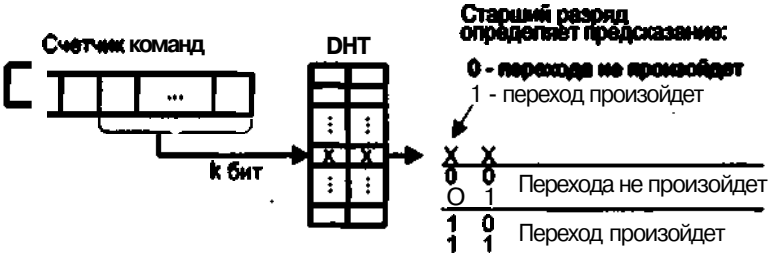


Рис. 9.28. Одноуровневая схема предсказания с таблицей DHT

Для обозначения таблицы истории переходов в данной схеме используют аббревиатуру DHT (Decode History Table). Слово «decoder»декодирование) в названии отражает особенность работы с таблицей. Если к обычной BHT обращение происходит при выборке любой команды, вне зависимости от того, на самом ли деле она команда условного перехода, поиск в DHT начинается только после декодирования команды, то есть когда выяснилось, что данная команда является командой условного перехода. Реализуется DHT на базе обычного ЗУ с произвольным доступом.

Последняя из обсуждаемых одноуровневых схем предсказания известна как *схема Смита* или *бимодальный предиктор*. *Отличие от предыдущего варианта выражается* лишь в способе реализации BHT (в схеме Смита сохранено именно это название таблицы). Таблица истории переходов организуется на базе кэш-памяти с ассоциативным отображением (рис. 9.29).

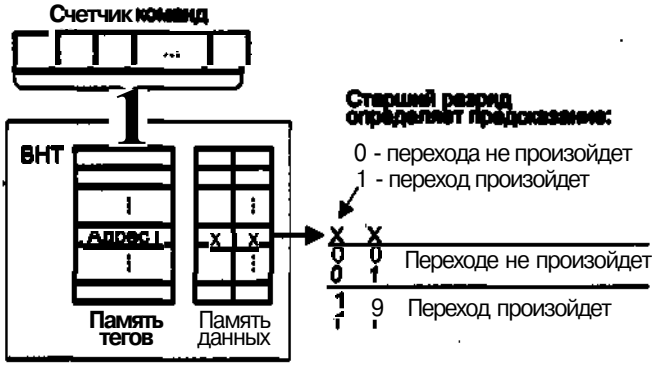


Рис. 9.29. Схема с таблицей истории переходов

В качестве ассоциативного признака (тега) при поиске нужного счетчика выступает адрес команды условного перехода. Такой подход позволяет ускорить по-

иск нужного счетчика и устраняет эффект наложения, но связан со значительными аппаратными затратами.

Результаты моделирования бимодальной схемы с двухразрядными счетчиками показаны на рис. 9.30. По этим данным среднюю точность предсказания можно оценить как 92,6!%.

Рис. 9.30. Точность предсказания двухразрядной бимодальной схемы [197]

В экспериментах, где данная идея проверялась на программах тестового пакета SPEC_95, средняя точность предсказания по всем программам пакета составила 53,9%. Несмотря на это, бимодальная схема с двухразрядными счетчиками довольно распространена На ее основе построены схемы предсказания переходов в процессорах Alpha 21164, R10000, PowerPC 620, UltraSPARC и др.

В заключение отметим, что во многих одноуровневых решениях таблица истории переходов (DHT или BHT) совмещена с буфером адреса перехода (BTB), что позволяет сэкономить на вычислении исполнительных адресов точек перехода.

Двухуровневые схемы предсказания переходов. Одноуровневые схемы предсказания ориентированы на те команды У П, очередной исход которых существенно зависит от их собственных предыдущих исходов. В то же время для многих команд программы наблюдается сильная зависимость не от собственных исходов, а от результатов выполнения других предшествующих им команд УП. Это обстоятельство призваны учесть *двухуровневые адаптивные схемы* предсказания переходов, впервые предложенные в [229]. Такие схемы часто называют *коррели*- условного перехода.

В коррелированных схемах предсказания переходов выделяются два уровня таблиц. В роли таблицы первого уровня может выступать регистр глобальной истории (GHR), и тогда двухуровневую схему предсказания называют *глобальной*. В качестве таблицы первого уровня может также быть взят массив регистров локальной истории (LHR), где отдельный регистр отражает последовательность последних исходов одной команды УП. Такая схема предсказания носит название *локальной*. Каждый элемент таблицы второго уровня служит для хранения истории переходов отдельной команды УП. Таблица второго уровня обычно состоит из двухразрядных счетчиков и организована в виде матрицы (рис. 9.31). Содержимое счетчика команд (адрес команды УП) определяет один из регистров в таблице первого и одну строку в таблице второго уровня. В свою очередь, кодовая комбинация (шаблон), хранящаяся в выбранном регистре таблицы первого уровня, определяет нужный счетчик в указанном ряду таблицы второго уровня. Выбранный счетчик используется для формирования предсказания. После выполнения команды содержимое регистра и счетчика обновляется.

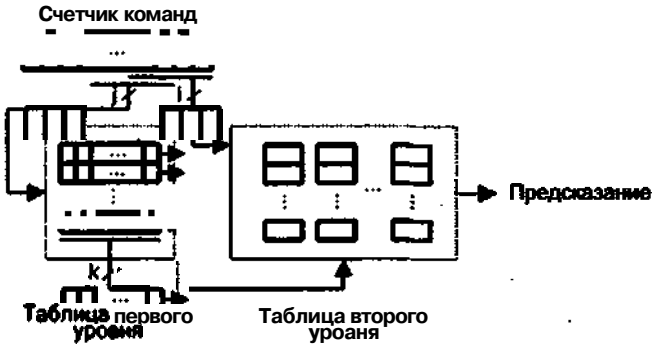


Рис. 9.31. Общая структура двухуровневой схемы предсказания переходов

Из описания логики двухуровневого предсказания следует, что выбор нужного счетчика обусловливается двумя источниками — адресом команды, для которой делается предсказание, и шаблоном, отражающим историю предшествующих переходов. Того же эффекта можно добиться при помощи схемы двухразрядного бимодального предиктора, если для доступа к **BHT** использовать шаблон, сформированный путем сложения по модулю 2, как это показано на рис. 9.23. Такая схема известна под названием *gshare*.

Гибридные схемы предсказания переходов. Для всех ранее рассмотренных стратегий характерна сильная зависимость точности предсказания от особенностей программ, в рамках которых эти стратегии реализуются. Та же самая схема, прекрасно проявляя себя с одними программными продуктами, с другими может давать совершенно неудовлетворительные результаты. Кроме того, необходимо учитывать еще один фактор. Прежде уже отмечалось, что точность предсказания повышается с увеличением глубины предыстории переходов, но происходит это лишь после накопления соответствующей информации, на что требуется определенное время: Период накопления предыстории принято называть *временем «разогрева»*. Пока идет «разогрев», точность предсказания весьма низка. Иными словами, ни одна из элементарных стратегий предсказания переходов не является универсальной - со всех сторон лучшей в любых ситуациях. *Гибридные* или *соревновательные схемы* объединяют в себе несколько различных механизмов предсказания — элементарных предикторов. Идея состоит в том, чтобы в каждой конкретной ситуации задействовать тот элементарный предиктор, от которого в данном случае можно ожидать наибольшей точности предсказания.

Гибридная схема предсказания переходов, предложенная Макфарлингом [165], содержит два элементарных предиктора, отличающихся по своим характеристикам (размером таблиц предыстории и временем «разогрева») и работающих независимо друг от друга. Выбор предиктора, наиболее подходящего в данной ситуации, обеспечивается селектором, представляющим собой таблицу двухразрядных счетчиков, которые часто называют *счетчиками выбора предиктора* (рис. 9.32),

Адресация конкретного счетчика в таблице (индексирование) осуществляется *k* младшими разрядами адреса команды условного перехода, для которой осуществляется предсказание. Обновление таблиц истории в каждом из предикторов производится обычным образом, как это происходит при их автономном исполь-

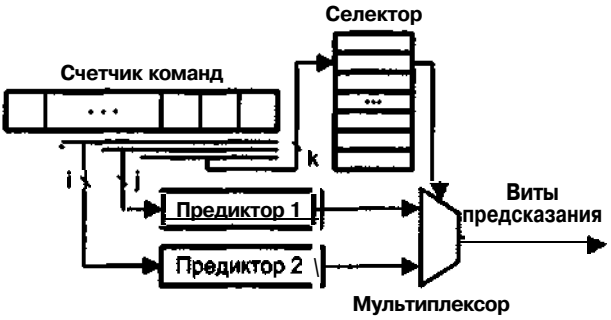


Рис. 9.32. Гибридный предиктор Макфарлинга

зовании. В свою очередь, изменение состояния счетчиков селектора выполняется по следующим правилам. Если оба предиктора одновременно дали одинаковое предсказание (верное или неверное), содержимое счетчика не изменяется. При правильном предсказании от первого предиктора и неверном от второго содержимое счетчика увеличивается, а в противоположном случае — уменьшается на единицу. Выбор предиктора, на основании которого делается результирующая оценка, реализуется с помощью мультиплексора, управляемого старшим разрядом соответствующего счетчика селектора.

В работе [95] идея гибридного механизма была обобщена на случай *n* предикторов. Общая структура такой схемы предсказания переходов показана на рис. 9.33. При выполнении команды УП предсказания формируются одновременно всеми предикторами, однако реальные действия осуществляются на основании только одного из них.

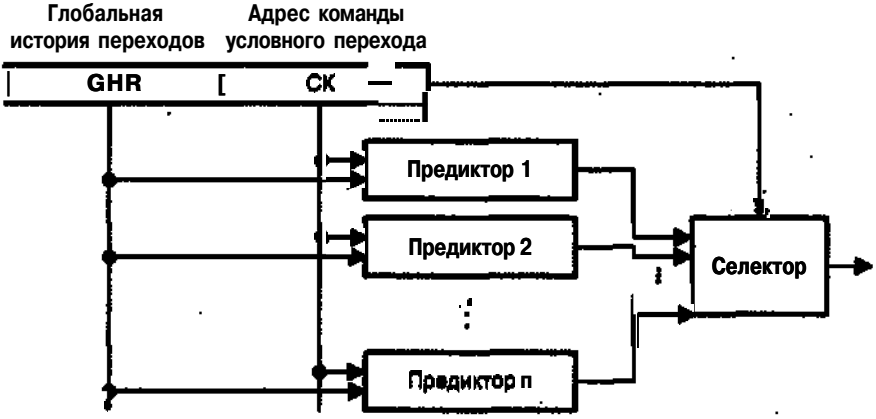


Рис. 9.33. Общая схема гибридного предиктора

Выбор подходящего предиктора обеспечивает механизм селекции (рис. 9.34). В схеме имеется буфер предыстории переходов (BTB), в котором каждая запись дополнена *n* двухразрядными счетчиками выбора предиктора, по числу используемых элементарных предикторов. Счетчики позволяют отследить самый предпочтительный элементарный предиктор для каждой команды У П, представленной в BTB.

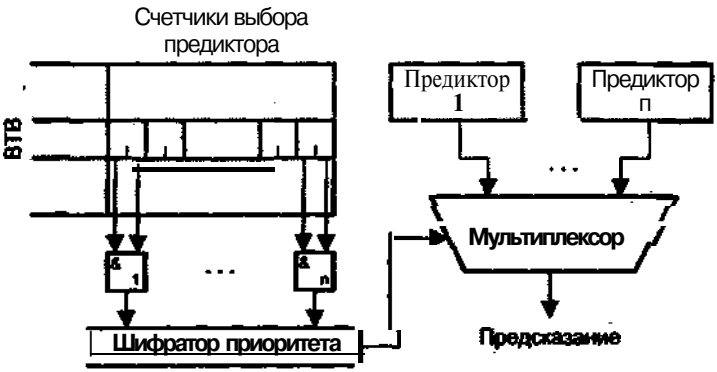


Рис. 9.34. Механизм выбора предиктора

При записи в **BTB** нового элемента во все ассоциированные с ним счетчики заносится число 3. Для каждой команды условного перехода предсказание генерируется всеми *n* предикторами, но во внимание принимаются только те из них, для которых соответствующий счетчик выбора предиктора содержит число 3. Если это число встретилось более чем в одном счетчике, выбор единственного предиктора, на основании которого и делается окончательное предсказание, обеспечивает шифратор приоритета. После выполнения команды условного перехода содержимое соответствующих ей счетчиков выбора обновляется, при этом действует следующий алгоритм. Если среди предикторов, счетчики которых равнялись 3, хотя бы один дал верное предсказание, то содержимое всех счетчиков, связанных с неверно сработавшими предикторами, уменьшается на единицу. В противном случае содержимое всех счетчиков, связанных с предикторами, прогноз которых подтвердился, увеличивается на единицу. Такая политика гарантирует, что по крайней мере в одном из счетчиков будет число 3. Еще одно преимущество рассматриваемой схемы выбора состоит в том, что она позволяет, например, отличить предиктор-<юракул», давший правильное предсказание последние пять раз, от предиктора, верно определившего исход последние четыре раза. Стандартные счетчики с насыщением такую дифференциацию не обеспечивают.

По имеющимся оценкам, точность предсказания переходов с помощью гибридных стратегий в среднем составляет 97,13%, что существенно выше по сравнению с прочими вариантами.

Асимметричная схема предсказания переходов. Асимметричная схема сочетает в себе черты гибридных и коррелированных схем предсказания. От гибридных схем она переняла одновременное срабатывание нескольких различных элементарных предикторов, В асимметричной схеме таких предикторов три, и каждый из них использует собственную таблицу PHT. Для доступа к таблицам, аналогично коррелированным схемам, используется как адрес команды условного перехода, так и содержимое регистра глобальной истории (рис. 9.35).

Шаблон для обращения к каждой из трех PHT формируется по-разному (применены различные функции хэширования). При выполнении команды условного перехода каждый из трех предикторов выдвигает *свое* предположение, но окончательное решение принимается по мажоритарной схеме. После завершения команды условного перехода содержимое всех трех таблиц обновляется.

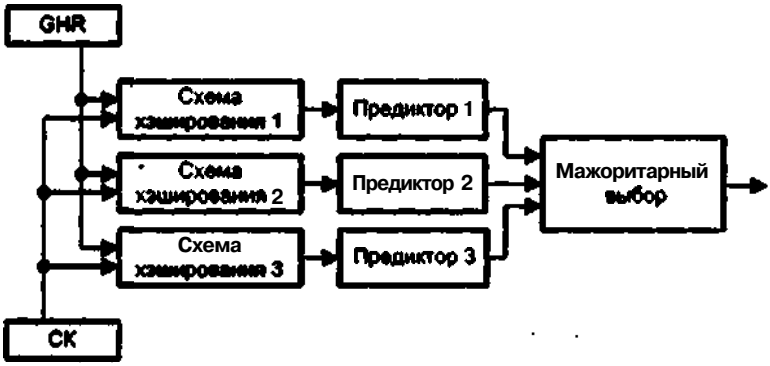


Рис. 9.35. Структура асимметричной схемы предсказания переходов

Правильный подбор алгоритмов хэширования позволяет практически исключить влияние на точность предсказания эффекта наложения. Средняя точность предсказания с помощью асимметричной схемы, полученная на тестовом пакете SPEC_95, составила 72,6%.

Суперконвейерные процессоры

Эффективность конвейера находится в прямой зависимости от того, с какой частотой на его вход подаются объекты обработки. Добиться n-кратного увеличения темпа работы конвейера можно двумя путями:

- разбиением каждой ступени конвейера на *n* «подступеней» при одновременном повышении тактовой частоты внутри конвейера также в *n* раз;
- включением в состав процессора *n* конвейеров, работающих с перекрытием.

В данном разделе рассматривается первый из этих подходов, известный как *суперконвейеризация* (термин впервые был применен в 1988 году). Иллюстрацией эффекта суперконвейеризации может служить диаграмма, приведенная на рис. 9.36, где рассмотрен ранее обсуждавшийся пример (см. рис. 9.3). Каждая из шести ступеней стандартного конвейера разбита на две более простые подступени, обозначенные индексами 1 и 2. Выполнение операции в подступенях занимает половину тактового периода. Тактирование операций внутри конвейера производится с частотой, вдвое превышающей частоту «внешнего» тактирования конвейера, благодарящему на каждой ступени конвейера можно в пределах одного «внешнего» тактового периода выполнить две команды.

В сущности, суперконвейеризация сводится к увеличению количества ступеней конвейера как за счет добавления новых ступеней, так и путем дробления имеющихся ступеней на несколько простых подступеней. Главное требование — возможность реализации операции в каждой подступени наиболее простыми техническими средствами, а значит, с минимальными затратами времени. Вторым, не менее важным, условием является одинаковость задержки во всех подступенях.

Критерием для причисления процессора к суперконвейерным служит число ступеней в конвейере команд. К суперконвейерным относят процессоры, где таких ступеней больше шести. Первым серийным суперконвейерным процессором считается MIPS R4000, конвейер команд которого включает в себя восемь ступе-

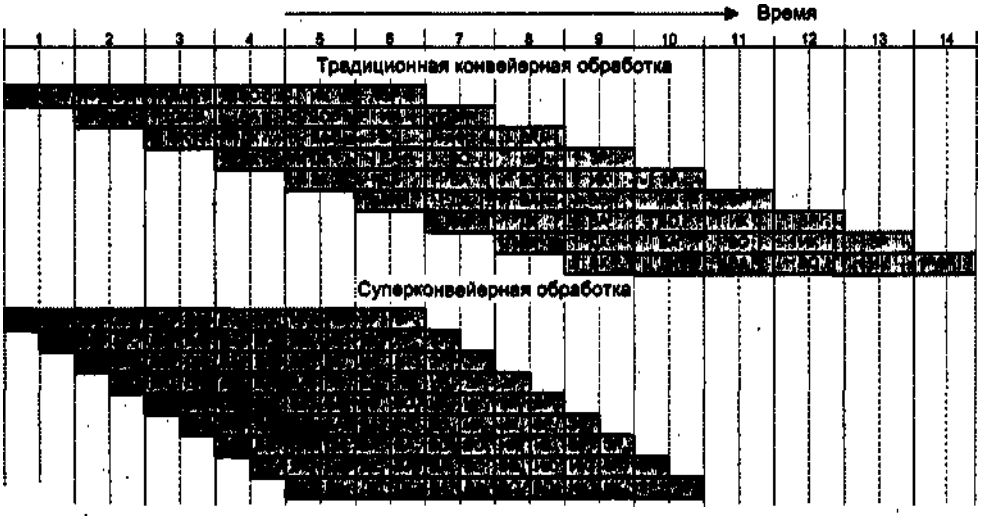


Рис. 9.36. Традиционная и суперконвейерная обработка команд

ней. Супер конвейеризация здесь стала следствием разбиения этапов выборки команды и выборки операнда, а также введения в конвейер дополнительного этапа проверки тега, появление которой обусловлено архитектурой системы команд машины.

Таблица 9.1 . Длина конвейера **команд** в **популярных** микропроцессорах

Тип микропроцессора	Количество етупаиай в конвейере команд
MIPS R4400	8
UltraSPARC I	9
Pentium III	10
Itanium	10
UltraSPARC III	14
Pentium 4	20

К сожалению, выигрыш, достигаемый за счет суперконвейеризации, на практике может оказаться лишь умозрительным. Удлинение конвейера ведет не только к усугублению проблем, характерных для любого конвейера, но и возникновению дополнительных сложностей. В длинном конвейере возрастает вероятность конфликтов. Дороже встает ошибка предсказания перехода - приходится очищать большее число ступеней конвейера, на что требуется больше времени. Усложняется логика взаимодействия ступеней конвейера. Тем не менее создателям ВМ удается успешно справляться с большинством из перечисленных проблем, свидетельством чего служит неуклонное возрастание числа ступеней в конвейерах команд современных процессоров (табл. 9.1).

Архитектуры с полным и сокращенным набором команд

Современная технология программирования ориентирована на языки высокого уровня (ЯВУ), главная задача которых — облегчить процесс написания программ. Более 90% всего процесса программирования осуществляют на ЯВУ. К сожалению, операции, характерные для ЯВУ, отличаются от операций, реализуемых машинными командами. Эта проблема получила название *семантического разрыва* и ведет она к недостаточно эффективному выполнению программ. Пытаясь преодолеть семантический разрыв, разработчики ВМ расширяют систему команд, дополняя ее командами, реализующими сложные операторы ЯВУ на аппаратурном уровне, вводят дополнительные виды адресации и т. п. Вычислительные машины, где реализованы эти средства, принято называть ВМ *с полным набором команд* (CISC — Complex Instruction Set Computer). К типу CISC можно отнести практически все ВМ, выпускавшиеся до середины 80-х годов и значительную часть из выпускаемых в настоящее время.

Характерные для CISC способы решения проблемы семантического разрыва, вместе с тем ведут к усложнению архитектуры ВМ, главным образом устройства управления, что, в свою очередь, негативно сказывается на производительности в целом. Кроме того, в CISC очень сложно организовать эффективный конвейер команд, который, как уже отмечалось, является одним из наиболее перспективных путей повышения производительности ВМ. Все это заставило более внимательно проанализировать программы, получаемые после компиляции с ЯВУ. Был предпринят комплекс исследований [128,158,177,178,209], в результате которых обнаружились интересные закономерности:

- Реализация сложных команд, эквивалентных операторам ЯВУ, требует увеличения емкости управляющей памяти в микропрограммном УУ. Микропрограм- как их доля в общем объеме программы зачастую не превышает 0,2%.

- В откомпилированной программе операторы ЯВУ реализуются в виде процедур (подпрограмм), поэтому на операции вызова процедуры и возврата из нее приходится от 15 до 45% вычислительной нагрузки.
- При вызове процедуры вызывающая программа передает этой процедуре некоторое количество аргументов. Согласно [209], в 98% случаев число передаваемых аргументов не превышает шести. Примерно такое же положение сложилось и с параметрами, которые процедура возвращает вызывающей программе. Более 80% переменных, используемых программой [177,178], являются локальными, то есть создаются при входе в процедуру и уничтожаются при выходе из нее. Количество локальных переменных, создаваемых отдельной процедурой, в 92% случаев не превышает шести [209].
- Почти половину операций в ходе вычислений составляет операция присваивания, сводящаяся к пересылке данных между регистрами, ячейками памяти или регистрами и памятью.

Детальный анализ результатов исследований привел к серьезному пересмотру традиционных архитектурных решений, следствием чего стало появление *архитектуры с сокращенным набором команд* (RISC - Reduced Instruction Set Computer) . Термин «RISC» впервые был использован Паттерсоном и Дитцелем в 1980 году.

Основные черты RISC-архитектуры

Главные усилия в архитектуре RISC направлены на построение максимально эффективного конвейера команд, то есть такого, где все команды извлекаются из памяти и поступают в ЦП на обработку в виде равномерного потока, причем ни одна команда не должна находиться в состоянии ожидания, а ЦП должен оставаться загруженным на протяжении всего времени. Кроме того, идеальным будет вариант, когда любой этап цикла команды выполняется в течение одного тактового периода,

Последнее условие относительно просто можно реализовать для этапа выборки. Необходимо лишь, чтобы все команды имели стандартную длину, равную ширине шины данных, соединяющей ЦП и память. Унификация времени исполнения для различных команд — значительно более сложная задача, поскольку наряду с регистровыми существуют также команды с обращением к памяти.

Помимо одинаковой длины команд, важно иметь относительно простую подсистему декодирования и управления: сложное устройство управления (УУ) будет вносить дополнительные задержки в формирование сигналов управления. Очевидный путь существенного упрощения УУ — сокращение числа выполняемых команд, форматов команд и данных, а также видов адресации.

Излишне напоминать, что в сокращенном списке команд должны оставаться те, которые используются наиболее часто. Исследования показали, что 80-90% времени выполнения типовых программ приходится на относительно малую часть команд (10-20%). К наиболее часто востребуемым действиям относятся пересылка данных, арифметические и логические операции. Основная причина, препятствующая сведению всех этапов цикла команды к одному тактовому периоду, - потенциальная необходимость доступа к памяти для выборки операндов и/или записи результатов. Следует максимально сократить число команд, имеющих доступ *к* памяти. Это соображение добавляет к ранее упомянутым принципам RISC еще два

- доступ к памяти во время исполнения осуществляется только командами «Чтение» и «Запись»;
- все операции, кроме «Чтение» и «Запись», имеют тип «регистр-регистр».

Для упрощения выполнения большинства команд и приведения их к типу «регистр-регистр* требуется снабдить ЦП значительным числом регистров общего назначения. Большое число регистров в регистровом файле ЦП позволяет обеспечить временное хранение промежуточных результатов, используемых как операнды в последующих операциях, и ведет к уменьшению числа обращений к памяти, ускоряя выполнение операций. Минимальное число регистров, равное 32, принято как стандарт де-факто большинством производителей RISC-компьютеров,

Суммируя сказанное, концепцию RISC-компьютера можно свести к следующим положениям:

- выполнение всех (или, по крайней мере, 75% команд) за один цикл;
- стандартная однословная длина всех команд, равная естественной длине слова и ширине шины данных и допускающая унифицированную поточную обработку всех команд;

- малое число команд (не более 128);
- малое количество форматов команд (не более 4);
- малое число способов адресации (не более 4);
- доступ к памяти только посредством команд «Чтение» и -«Запись»;
- все команды, за исключением «Чтения» и «Записи», используют внутрипроцессорные межрегистровые пересылки;
- устройство управления «жесткой» логикой;
- относительно большой (не менее 32) процессорный файл регистров общего назначения (согласно [210] число РОН в современных RISC-микропроцессорах может превышать 500).

Регистры в RISC-процессорах

Отличительная черта RISC-архитектуры - большое число регистров общего назначения, что объясняется стремлением свести все пересылки к типу «регистр-регистр». Но увеличение числа РОН способно дать эффект лишь при разумном их использовании. Оптимизация использования регистров в RISC-процессорах обеспечивается как программными, так и аппаратными средствами.

Программная оптимизация выполняется на этапе компиляции программы, написанной на ЯВУ. Компилятор стремится так распределить регистры процессора, чтобы разместить в них те переменные, которые в течение заданного периода времени будут использоваться наиболее интенсивно.

На начальном этапе компилятор выделяет каждой переменной виртуальный регистр. Число виртуальных регистров в принципе не ограничено. Затем компилятор отображает виртуальные регистры на ограниченное количество физических регистров. Виртуальные регистры, использование которых не перекрывается, отображаются на один и тот же физический регистр. Если в определенном фрагменте программы физических регистров не хватает, то их роль для оставшихся виртуальных регистров выполняют ячейки памяти. В ходе вычислений содержимое каждой такой ячейки с помощью команды «Чтение» временно засылается в регистр, после чего командой «Запись» вновь возвращается в ячейку памяти.

Задача оптимизации состоит в определении того, каким переменным в данной точке программы выгоднее всего выделить физические регистры. Наиболее распространенный метод, применяемый для этой цели, известен *как раскраска графа*. В общем случае метод формулируется следующим образом. Имеется граф, состоящий из узлов и ребер. Необходимо раскрасить узлы так, чтобы соседние узлы имели разный цвет и чтобы при этом общее количество привлеченных цветов было минимальным. В нашем случае роль узлов выполняют виртуальные регистры. Если два виртуальных регистра одновременно присутствуют в одном и том же фрагменте программы, они соединяются ребром. Делается попытка раскрасить граф в n цветов, где n — число физических регистров. Если такая попытка не увенчалась успехом, то узлам, которые не удалось раскрасить, вместо физических регистров выделяются ячейки в памяти.

На рис. 9.37 приведен пример раскраски графа [36], в котором шесть виртуальных регистров отображаются на три физических. Показаны временная последо-

вательность активного вовлечения в выполнение каждого виртуального регистра (рис. 9.37, *а)* и раскрашенный граф (рис. 9.37, *б)*.

Как видно, не удалось раскрасить только виртуальный регистр F, его придется отображать на ячейку памяти.

Аппаратная оптимизация использования регистров в RISC-процессорах ориентирована на сокращение затрат времени при работе с процедурами. Наибольшее время в программах, написанных на ЯВУ, расходуется на вызовы процедур и возврат из них. Связано это с созданием и обработкой большого числа локальных переменных и констант. Одним из механизмов для борьбы с этим эффектом являются так называемые *регистровые окна*. Главная их задача — упростить и ускорить передачу параметров от вызывающей процедуры к вызываемой и обратно.

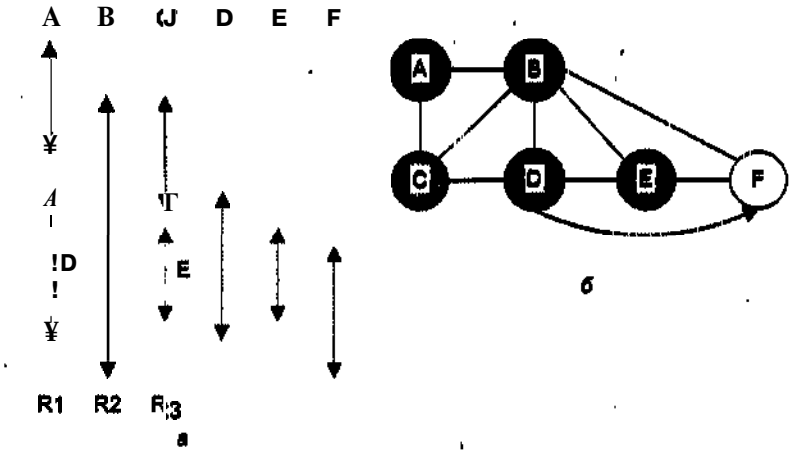


Рис. 9.37. Иллюстрация метода раскраски графа: *а* - временная последовательность активного использования виртуальных регистров; *б* - граф взаимного использования регистров

Регистровый файл разбивается на группы регистров, называемые окнами. Отдельное окно назначается глобальным переменным. Глобальные регистры доступны всем процедурам, выполняемым в системе в любое время. С другой стороны, каждой процедуре выделяется отдельное окно в регистровом файле. Все окна имеют одинаковый размер (обычно по 32 регистра) и состоят из трех полей. Левое поле каждого регистрового окна одновременно является и правым полем предшествующего ему окна (рис. 9.38). Среднее поле служит для хранения локальных переменных и констант процедуры.

База окна (первый в последовательности регистров окна) указывается полем, называемым указателем текущего окна (CWP, Current Window Pointer), обычно расположенным в регистре (слове) состояния ЦП. Если текущей процедуре назначено регистровое окно j CWP содержит значение j

Каждой вновь вызванной процедуре выделяется регистровое окно, непосредственно следующее за окном вызвавшей ее процедуры. Последние *k* регистров окна; одновременно являются первыми *k* регистрами окна j + 1. Если процедура, занимающая окно j обращается к процедуре, которой в данной архитектуре должно быть назначено окно j + 1, она может передать в процессе вызова *k* аргументов.

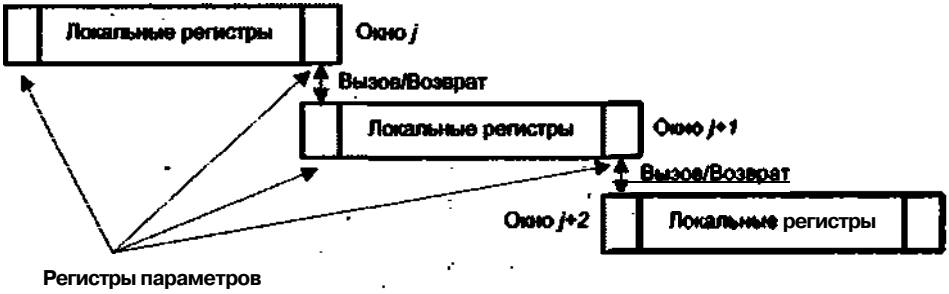


Рис. 9.38. Перекрытие регистровых окон

Упомянутые *k* регистров сразу же будут доступны вызванной процедуре без всяких пересылок. Естественно, вызов приведет к увеличению содержимого поля CPW на единицу.

Глубина вложения процедур одна в другую может быть весьма велика, и желательно, чтобы количество регистровых окон не было сдерживающим фактором. Это достигается за счет организации окон в виде циклического буфера.

На рис. 9.39 показан циклический буфер из шести окон, заполненный на глубину 4 (процедура A вызвала B, B вызвала C, C вызвала D). *Указатель текущего окна* (CWP) идентифицирует окно активной на данный момент процедуры - D, то есть окно 0_3. При выполнении процедуры все ссылки на регистры в командах преобразуются в смещение относительно C WP. *Указатель сохраненного окна* (SWP, Saved

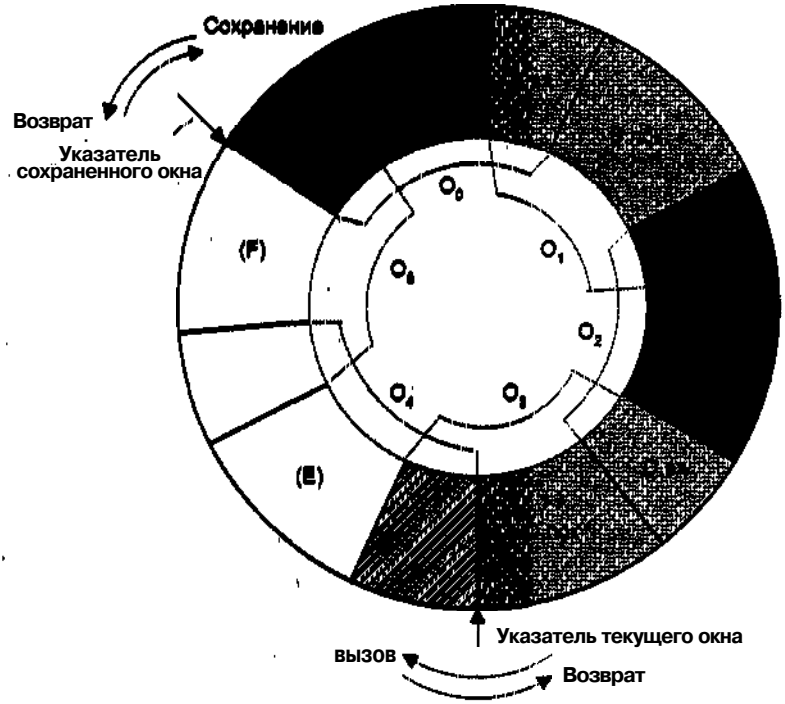


Рис. 9.39. Циклический буфер из пересекающихся регистровых окон

Window Pointer) содержит номер последнего из окон, сохраненных в памяти по причине переполнения циклического буфера. Если процедура D теперь вызовет процедуру E, аргументы для нее она поместит в общее для обеих поле регистровых окон (пересечение окон 0_3 и O_4 а значение CWP увеличится на единицу, то есть CWP будет показывать на окно 0_4.

Если далее процедура E вызовет нроцедуру F, то этот вызов при существующем состоянии буфера не может быть выполнен, поскольку окно для $F(0_5)$ перекрывается с окном процедуры A (0_0). Следовательно, при попытке F начать загружать правое поле своего окна будут потеряны параметры процедуры A ($A_{вх}$). Поэтому когда CWP увеличивается на единицу (операция выполняется по модулю 6) и оказывается равным SWP, возникает прерывание и окно процедуры A сохраняется в памяти (запоминаются только поля $A_{вх}$ и $A_{вых}$). Далее значение CWP инкрементируется и производится вызов процедуры F, Аналогичное прерывание происходит и при возврате, например когда выполнится возврат из B в A CWP уменьшает-приведет к восстановлению содержимого окна процедуры A из памяти.

Как видно из примера, регистровый файл из *n* окон способен поддерживать *n* - 1 вызов процедуры. Число *n* не должно быть большим. В [208] показано, что при 8 регистровых окнах сохранение и восстановление окон в памяти требуется лишь для 1% операций вызова процедур. В ВМ Pyramid, например, используется 16 окон по 32 регистра в каждом.

Теоретически такой прием не исключен и в CISC. Однако устройство управления CISC-процессора оккупирует на кристалле более 50% площади, оставляя мало места для других подсистем, в частности для большого файла регистров. У У RISC занимает порядка 10% поверхности кристалла, предоставляя возможность иметь большой регистровый файл.

Другая часто встречаемая техника аппаратной оптимизации использования регистров — придание некоторым из них специального качества: такие регистры в состоянии принимать на себя имя любого РОН.. Набор регистров, обладающих подобным свойством, называют *буфером переименования*. Прием оказывается очень удобным при конвейеризации команд и позволяет предотвратить конфликты, когда одна команда хочет воспользоваться регистром, в данный момент занятым другой командой.

Преимущества и недостатки RISC

Сравнивая достоинства и недостатки CISC и RISC, невозможно сделать однозначный вывод о неоспоримом преимуществе одной архитектуры над другой. Для отдельных сфер использования ВМ лучшей оказывается та или иная. Тем не менее ниже приводится основная аргументация «за» и «против» RISC-архитектуры,

Для технологии RISC характерна сравнительно простая структура устройства управления. Площадь, выделяемая на кристалле микросхемы для реализации УУ, существенно меньше.Так, в RISC I она составляет 6%,а в RISC II - 10%. Как следствие, появляется возможность разместить на кристалле большое число регистров ЦП (138 в RISC II). Кроме того, остается больше места для других узлов ЦП и для дополнительных устройств: кэш-памяти, блока арифметики с плавающей запятой, части основной памяти, блока управления памятью, портов ввода/вывода.

Унификация набора команд, ориентация на потоковую конвейерную обработку, унификация размера команд и длительности их выполнения, устранение пери-

одов ожидания в конвейере — все эти факторы положительно сказываются на общем быстродействии. Простое устройство управления имеет немного вентилей и, следовательно, короткие линии связи для прохождения сигналов управления. Малое число команд, форматов и режимов приводит к упрощению схемы декодирования, и оно происходит быстрее. Применяемое в RISC УУ с «жесткой»логикой быстрее микропрограммного. Высокой производительности способствует и упрощение передачи параметров между процедурами. Таким образом, применение RISC ведет к сокращению времени выполнения программы или увеличению скорости, за счет сокращения числа циклов на команду.

Простота У У, сопровождаемая снижением стоимости и повышением надежности, также говорит в пользу RISC. Разработка УУ занимает меньше времени. Простое УУ будет содержать меньше конструктивных ошибок и поэтому более надежно.

Многие современные CISC-машины, такие как VAX 11/780, VA X-8600, имеют много средств для прямой поддержки функций ЯВУ, наиболее частых в этих языках (управление процедурами, операции с массивами, проверка индексов массивов, защита информации, управление памятью и т. д.). RISC также обладает рядом средств для непосредственной поддержки ЯВУ и упрощения разработки компиляторов ЯВУ, благодаря чему эта архитектура в плане поддержки ЯВУ ни в чем не уступает CISC,

Недостатки RISC прямо связаны с некоторыми преимуществами этой архитектуры. Принципиальный недостаток - сокращенное число команд: на выполнение ряда функций приходится тратить несколько команд вместо одной в CISC. Это удлиняет код программы, увеличивает загрузку памяти и трафик команд между памятью и ЦП. Недавние исследования показали, что RISC-программа в среднем на 30% длиннее CISC-программы, реализующей те же функции.

Хотя большое число регистров дает существенные преимущества, само по себе оно усложняет схему декодирования номера регистра, тем самым увеличивается время доступа к регистрам,

УУ с «жесткой» логикой, реализованное в большинстве RISC-систем, менее гибко, более склонно к ошибкам, затрудняет поиск и исправление ошибок, уступает при выполнении сложных команд.

Однословная команда исключает прямую адресацию для полного 32-битового адреса Поэтому ряд производителей допускают небольшую часть команд двойной длины, например в Intel 80960.

Суперскалярные процессоры

Поскольку возможности по совершенствованию элементной базы уже практически исчерпаны, дальнейшее повышение производительности ВМ лежит в плоскости архитектурных решений. Как уже отмечалось, один из наиболее эффективных подходов в этом плане - введение в вычислительный процесс различных уровней параллелизма Ранее рассмотренный конвейер команд - типичный пример такого подхода Тем же целям служат и арифметические конвейеры, где конвейеризации подвергается процесс выполнения арифметических операций. Дополнительный уровень параллелизма реализуется в векторных и матричных процессорах, но только при обработке многокомпонентных операндов типа векторов и массивов. Здесь высокое быстродействие достигается за счет одновременной обработки всех компонентов вектора или массива, однако подобные операнды характерны лишь для

достаточно узкого круга решаемых задач. Основной объем вычислительной нагрузки обычно приходится на скалярные вычисления, то есть на обработку одиночных операндов, таких, например, как целые числа. Для подобных вычислений дополнительный параллелизм реализуется значительно сложнее, но тем не менее возможен и примером могут служить суперскалярные процессоры.

Суперскалярным (этот термин впервые был использован в 1987 году [45]) называется центральный процессор (ЦП), который одновременно выполняет более чем одну скалярную команду. Это достигается за счет включения в состав ЦП нескольких самостоятельных функциональных (исполнительных) блоков, каждый из которых отвечает за свой класс операций и может присутствовать в процессоре в нескольких экземплярах. Так, в микропроцессоре Pentium III блоки целочисленрах Pentium 4 и Athlon — троированы. Структура типичного суперскалярного процессора [234] показана на рис. 9.40, Процессор включает в себя шесть блоков: выборки команд, декодирования команд, диспетчеризации команд, распределения команд по функциональным блокам, блок исполнения и блок обновления состояния.

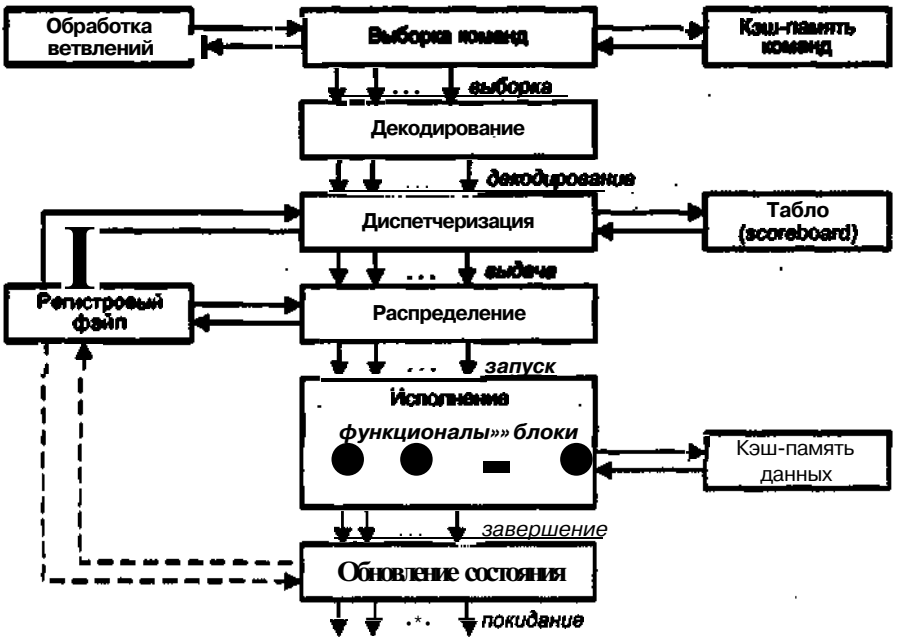


Рис. 0.40. Архитектура суперскалярного процессора

Блок выборки команд извлекает команды из основной памяти через кэш-память команд. Этот блок хранит несколько значений счетчика команд и обрабатывает команды условного перехода.

Блок декодирования расшифровывает код операции, содержащийся в извлеченных из кэш-памяти командах, В некоторых суперскалярных процессорах, например в микропроцессорах фирмы Intel, блоки выборки и декодирования совмещены.

Блоки диспетчеризации и распределения взаимодействуют между собой и в совокупности играют в суперскалярном процессоре роль контроллера трафика. Оба

блока хранят очереди декодированных команд. Очередь блока распределения часто рассредоточивается по несколько самостоятельным буферам — накопителям команд или схемам резервирования (reservation station), — предназначенным для хранения команд, которые уже декодированы, но еще не выполнены. Каждый накопитель команд связан со своим функциональным блоком (ФБ), поэтому число накопителей обычно равно числу ФБ, но если в процессоре используется несколько однотипных ФБ, то им придается общий накопитель. По отношению к блоку диспетчеризации накопители команд выступают в роли виртуальных функциональных устройств. Оба вида очередей показаны на рис. 9.41 [234]. В некоторых суперскалярных процессорах они объединены в единую очередь.

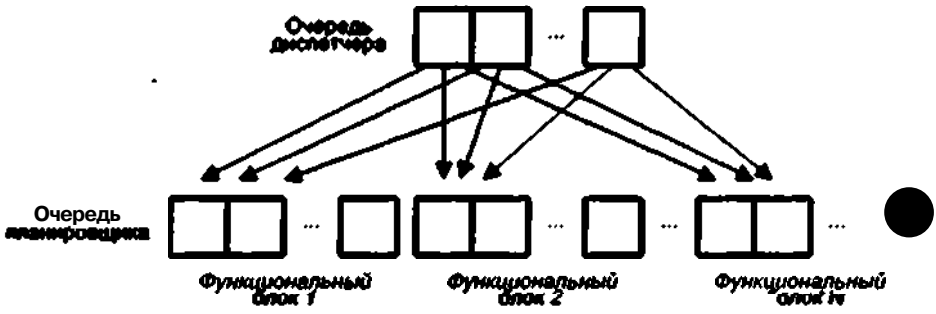


Рис. 9.41. Очереди диспетчеризации и распределения

В дополнение к очереди, блок диспетчеризации хранит также список свободных функциональных блоков, называемый *табло* (Scoreboard). Табло использует-диспетчеризации извлекает команды из своей очереди, считывает из памяти или регистров операнды этих команд, после чего, в зависимости от состояния табло, помещает команды и значения операндов в очередь распределения. Эта операция называется *выдачей команд.* Блок распределения в каждом цикле проверяет каждую команду в своих очередях на наличие всех необходимых для ее выполнения операндов и при положительном ответе начинает выполнение таких команд в соответствующем функциональном блоке.

Блок исполнения состоит из набора функциональных блоков. Примерами ФБ могут служить целочисленные операционные блоки, блоки умножения и сложения с плавающей запятой, блок доступа к памяти. Когда исполнение команды завершается, ее результат записывается и анализируется *блоком обновления состояния,* который обеспечивает учет полученного результата теми командами в очередях распределения, где этот результат выступает в качестве одного из операндов.

Как было отмечено ранее, суперскалярность предполагает параллельную работу максимального числа исполнительных блоков, что возможно лишь при одновременном выполнении нескольких скалярных команд. Последнее условие хорошо сочетается с конвейерной обработкой, при этом желательно, чтобы в суперскалярном процессоре было несколько конвейеров, например два или три.

Подобный подход реализован в микропроцессоре Intel Pentium, где имеются два конвейера, каждый со своим АЛУ (рис. 9.42). Отметим, что здесь, в отличие от стандартного конвейера, в каждом цикле необходимо производить выборку более

чем одной команды. Соответственно, память ВМ должна допускать одновременное считывание нескольких команд и операндов, что чаще всего обеспечивается за счет ее модульного построения.

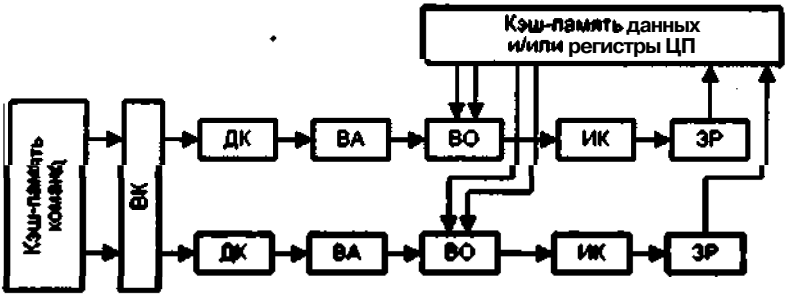


Рис. 9.42. Суперскалярный процессор с двумя конвейерами

Более интегрированный подход к построению суперскалярного конвейера показан на рис, 9.43. Здесь блок выборки (ВК) извлекает из памяти более одной команды и передает их через ступени декодирования команды и вычисления адресов операндов в блок выборки операндов (ВО). Когда операнды становятся доступными, команды распределяются по соответствующим исполнительным блокам. Обратим внимание, что операции «Чтение», «Запись» и «Переход» реализуются самостоятельными исполнительными блоками. Подобная форма суперскалярного процессора используется в микропроцессорах Pentium II и Pentium III фирмы Intel, а форма с тремя конвейерами - в микропроцессоре Athlon фирмы AMD,

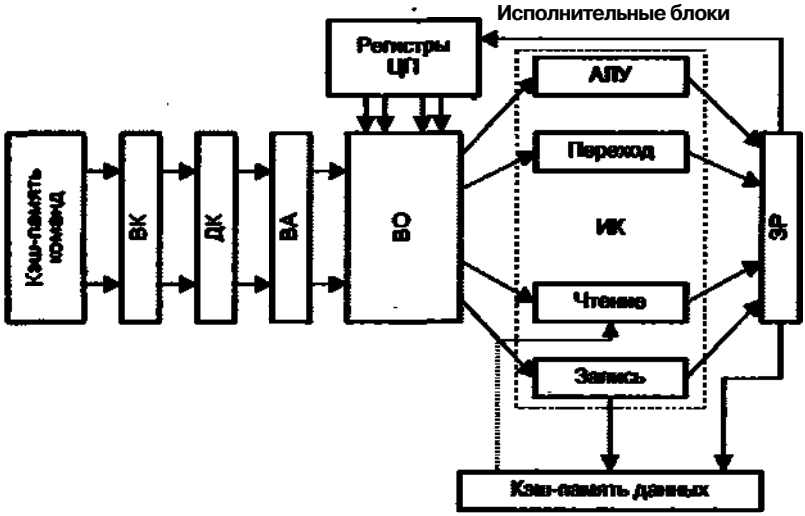


Рис. 9.43. Суперскалярный конвейер со специализированными исполнительными блоками

По разным оценкам, применение суперскалярного подхода приводит к повышению производительности ВМ в пределах от 1,8 до 8 раз.

Для сравнения эффективности суперскалярного и суперконвейерного режимов на рис. 9.44 показан процесс выполнения восьми последовательных скалярных команд. Верхняя диаграмма иллюстрирует суперскалярный конвейер, обеспечивающий в каждом тактовом периоде одновременную обработку двух команд. Отметим, что возможны суперскалярные конвейеры, где одновременно обрабатывается большее количество команд.

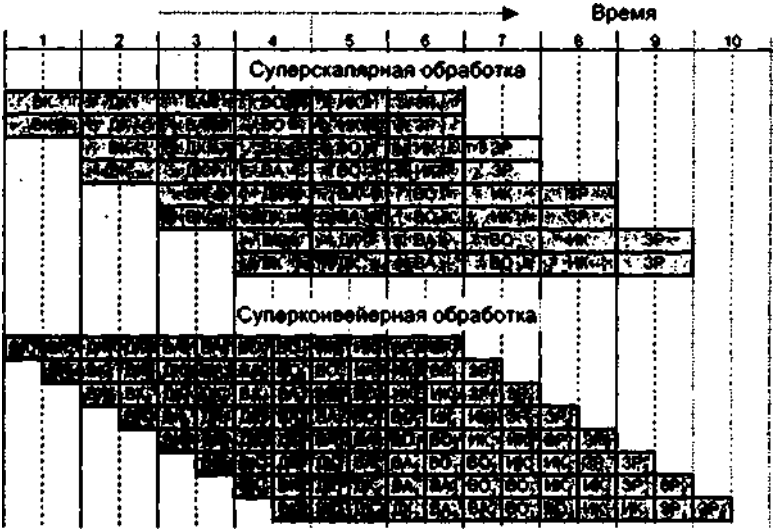


Рис. 9.44. Сравнение суперскалярного и сулерконвейерного подходов

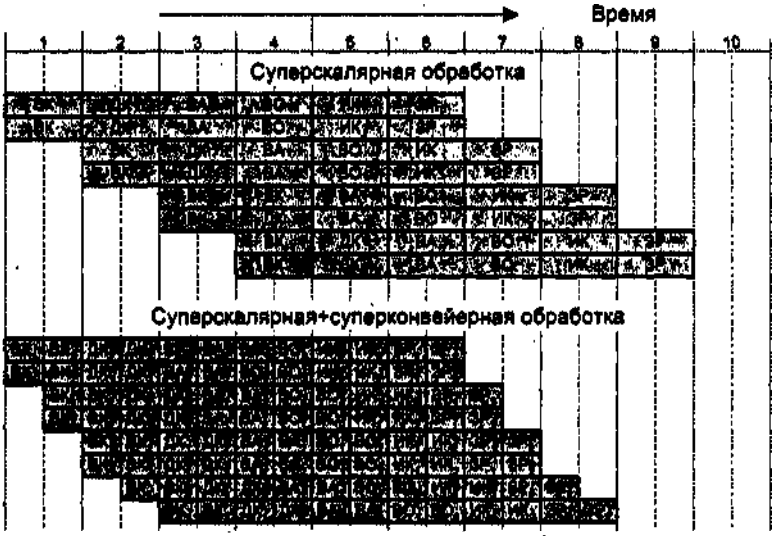


Рис. 9.45. Сравнение эффективности стандартной суперскалярной и совмещенной схем суперскалярных вычислений

В процессорах некоторых ВМ реализованы как суперскалярность, так и суперконвейеризация (рис. 9.45). Такое совмещение имеет место в микропроцессорах Athlon и Duron фирмы AMD, причем охватывает оно не только конвейер команд, но и блок обработки чисел в форме с плавающей запятой.

Особенности реализации суперскалярных процессоров

Поскольку в реальных суперскалярных процессорах множественность функциональных блоков сочетается с множественностью конвейеров команд, таким процессорам присущи все виды зависимостей, характерные для одинарных конвейеров, причем положение дел усугубляется тем, что конвейеров несколько. В суперскалярных процессорах одновременная работа нескольких-конвейеров становится источником дополнительных неувязок, в частности проблемы последовательности поступления команд на исполнение и проблемы последовательности завершения команд.

Первая из упомянутых проблем возникает, когда очередность выдачи декодированных команд на исполнительные блоки отличается от последовательности, предписанной программой. Подобная ситуация известна как *неупорядоченная выдача команд* (out-of-order issue). Термин *упорядоченная выдача команд* (in-order issue) применяют, когда команды покидают ступени, предшествующие ступени исполнения, в определенном программой порядке. В обоих случаях завершение команд обычно неупорядочено *(неупорядоченное завершение команд* - out-of-order completion), и это является второй проблемой. *Упорядоченное завершение* происходит реже. Например, в последовательности

MUL R1, R2, R3{R1 ← R2 × R3}
ADD R4, R5, R6 {R4 ← R5 + R6},

даже если команда умножения MUL поступит в исполнительный блок до команды сложения ADD, умножение может потребовать много циклов, из-за чего команда MUL будет завершена позже, чем ADD, В современных микропроцессорах каждая команда разбивается на простейшие микрооперации, которые далее выполняются суперскалярным процессорным ядром в порядке, который удобен процессору.

В суперскалярных процессорах, с их множественными конвейерами и неупорядоченными выдачами/завершениями, взаимозависимость команд представляет серьезную проблему. Кроме того, существует еще один фактор, характерный только для суперскалярных процессоров, - конфликт по функциональному блоку, когда на него претендуют несколько команд, поступивших из разных конвейеров.

Пусть имеется последовательность

ADD R1, R2, R3(R1 ←R2 + R3}
SUB R4, R5, R6 {R4 ←R5 - R6}.

Зависимости между командами здесь нет, однако если в ЦП имеется только одно АЛУ, одновременное выполнение указанных операций невозможно.

Стратегии выдачи и завершения команд

В режиме параллельного выполнения нескольких команд процессор должен определить, в какой очередности ему следует:

выбирать команды из памяти;

-выполнять эти команды;
-позволять командам изменять содержимое регистров и ячеек памяти.

Для достижения максимальной загрузки всех ступеней своих конвейеров суперскалярный процессор должен варьировать все перечисленные виды последовательностей, но так, чтобы получаемый результат был идентичен результату при выполнении команд в порядке, определенном программой. Значит, процессор обязан учитывать все виды зависимостей и конфликтов.

В самом общем виде стратегии выдачи и завершения команд можно сгруппировать в такие категории:

- упорядоченная выдача и упорядоченное завершение;
- упорядоченная выдача и неупорядоченное завершение;
- неупорядоченная выдача и неупорядоченное завершение.

Проанализируем каждый из этих вариантов на примере суперскалярного процессора с двумя конвейерами [200]. Процессор способен одновременно выбирать и декодировать две команды, причем передача обеих команд на декодирование должна также производиться одновременно. В состав процессора входят три отдельных функциональных блока (ФБ) и два устройства, обеспечивающие запись результата В рассматриваемом примере предполагается существование следующих ограничений на выполнение программного кода из шести команд (I1-16):

- I1 требует для своего выполнения двух циклов процессора;
- I3 и I4 имеют конфликт за обладание одним и тем же ФБ;
- I5 зависит от значения, вычисляемого командой I4;
-I5 и I6 конфликтуют за обладание одним и тем же ФБ.

Упорядоченная выдача и **упорядоченное завершение.** Наиболее простым в реализации вариантом является выдача декодированных команд на исполнение в том порядке, в котором они должны выполняться *по* программе (упорядоченная выдача), с сохранением той же последовательности записи результатов (упорядоченное завершение). Хотя такая стратегия и применялась в первых процессорах типа Pentium, сейчас она практически не встречается. Тем не менее ее обычно берут в качестве точки отсчета при сравнении различных стратегий выдачи и завершения. Согласно данному принципу, все что затрудняет завершение команды в одном конвейере, останавливает и другой конвейер, так как команды должны покидать конвейеры, соответствуя порядку поступления на них. Пример использования подобной стратегии показан на рис. 9.46.

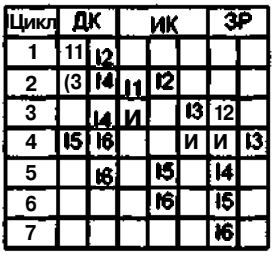
Здесь производятся одновременная выборка и декодирование двух команд. Чтобы принять очередные команды, процессор должен ожидать, пока освободятся обе части ступени декодирования. Для упорядочивания завершения выдача команд приостанавливается, если возникает конфликт за общий функциональный блок или если функциональному блоку для формирования результата требуется более чем один такт процессора.

В рассматриваемом примере время задержки от декодирования первой команды до записи последнего результата составляет 8 тактов.

Упорядоченная выдача и неупорядоченное завершение. Стратегии с неупорядоченным завершением дают возможность одному из конвейеров продолжать

Рис. 9.46. Упорядоченная выдача и упорядоченное завершение

работать при «заторе» в другом, при этом команды, стоящие в программе «позже», могут быть фактически выполнены раньше предыдущих, «застрявших» в другом конвейере. Естественно, процессор, должен гарантировать, что результаты не будут записаны в память, а регистры не будут модифицироваться в неправильной последовательности, поскольку при этом могут получиться ошибочные результаты.



Рис, 9.47. Упорядоченная выдача и неупорядоченное завершение

Проиллюстрируем стратегию с упорядоченной выдачей и неупорядоченным завершением (рис. 9.47). При заданных ей условиях допускается, что команда I2 может быть завершена еще до окончания исполнения команды И. Это позволяет команде I3 завершиться на один такт раньше, вследствие чего результаты выполнения команд И и I3 записываются в одном и том же такте. При неупорядоченной выдаче в любой момент времени в стадии исполнения может находиться любое число команд, а степень параллелизма ограничена только числом функциональных блоков. По сравнению с предыдущей стратегией, возможность неупорядоченного завершения команд привела к сокращению времени выполнения шести команд на один цикл процессора.

Неупорядоченная выдача и неупорядоченное завершение. Неупорядоченная выдача развивает предыдущую концепцию, разрешая процессору нарушать предписанный программой порядок выдачи команд на исполнение. Чтобы обеспечить неупорядоченную выдачу команд, в конвейере необходимо максимально развязать ступени декодирования и исполнения. Это обеспечивается с помощью буферной памяти, называемой *окном команд.* Каждая декодированная команда сначала помещается в окно команд. Процессор может продолжать выборку и декодирование новых команд вплоть до полного заполнения буфера. Выдача команд из буфера на исполнение определяется не последовательностью их поступления, а мерой

готовности. Иными словами, любая команда, для которой уже известны значения всех операндов, при условии что функциональный блок, требуемый для ее исполнения, свободен, немедленно выдается из буфера на исполнение.

Цикл	ДК		Окно	ИК			ЗР	
1	I1	I2						
2	I3	I4	I1, I2	I1	I2			
3	I5	I6	I3, I4	И		I3	I2	
4			I4, I5, I6		I6	I4	И	I3
5			I5		I5		I4	I6
6							I5	

Рис. 9.48. Неупорядоченная выдача и неупорядоченное завершение

Стратегию иллюстрирует рис. 9.48. В каждом цикле процессора две команды из ступени декодирования пересылаются в окно команд (с учетом ограничения на размер буфера). Выдача команд из буфера производится по мере их готовности. Так, в рассматриваемом примере возможна выдача команды I6 до выдачи команды I5 (напомним, что I5 зависит от I4, а I6 - нет). Таким образом, сберегается один такт, как в ступени исполнения команды (ИК), так и в ступени записи результата (ЗР), и сквозная экономия по сравнению с рис. 9.47 составляет один цикл процессора На рисунке изображено окно команд, но оно не является дополнительной ступенью конвейера.

Стратегии неупорядоченной выдачи и неупорядоченного завершения также свойственны ранее рассмотренные ограничения. Команда не может быть выдана, если она приводит к зависимости или конфликту. Разница заключается в том, что к выдаче готово больше команд, и это позволяет уменьшить вероятность приостановки конвейера.

Аппаратная поддержка суперскалярных операций

Из предыдущих рассуждений следует, что неупорядоченные выдача и завершение команд - это дополнительный потенциал повышения производительности суперскалярного процессора, для реализации которого, вместе с тем, необходимо решить две проблемы:

- устранить зависимость команд по данным (речь идет о зависимостях типа ЧПЗ и ЗПЗ), то есть исключить использование в качестве операнда «устаревшего» значения регистра и не допускать, чтобы очередная команда программы из-за нарушения последовательности выполнения команд занесла свой результат в регистр еще до того, так это сделала предшествующая команда;
- сохранить такой порядок выполнения команд, чтобы общий итог вычислений остался идентичным результату, получаемому при строгом соблюдении программной последовательности.

Несмотря на то что обе задачи в принципе могут быть решены чисто программными средствами еще на этапе компиляции программы, в реальных суперскалярных процессорах для этих целей имеются соответствующие аппаратные средства. Каждая из перечисленных проблем решается своими методами и своими аппарат-

ными средствами. Для устранения зависимости по данным используется прием, известный *как переименование регистров*. Способ решения второй проблемы обобщенно называют *переупорядочиванием команд* или *откладыванием исполнения команд*.

Переименование регистров

Когда команды выдаются и завершаются упорядоченно, каждый регистр в любой точке программы содержит именно то значение, которое диктуется программой. Применение стратегий с неупорядоченной выдачей и завершением команд ведет к тому, что запись в регистры может происходить также неупорядоченно и отдельные команды, обратившись к какому-то регистру, вместо нужного получат «устаревшее» или «опережающее» значение. Пусть имеется последовательность команд:

```
I1:MUL R2, RO, R1{R2 ← R0 x RI} .
I2: ADD RO, R1, R2{R0 ← R1 + R2}
I3: SUB R2, R0, R1 {R2 ← R0 – R1}.
```

При неупорядоченных выдаче/завершении возможны ситуации, приводящие к неверному результату, например:

- команда I2 была исполнена до того, как И успела записать в регистр R2 свой результат, то есть I2 использовала «старое» содержимое R2;
- команда I3 исполнена раньше, чем И, в результате чего неверный результат будет получен в I2, а по завершении цепочки из трех команд в R2 останется результат И вместо результата I3.

Ясно, что здесь нарушение порядка выполнения команд ведет к неправильному результату. Вводя новые регистры R O a и R2a, получим иную последовательность:

```
I1: MUL R2, RO, R1{R2 ← R0 × RI}
I2: ADD ROa, R1, R2{ROa ← R1 + R2}
I3: SUB R2a, ROa, RI {R2a ← ROa - R1},
```

где возможность конфликта устранена. Такой метод известен как *переименование регистров* (register renaming).,

Основная идея переименования регистров состоит в том, что каждый новый результат записывается в один из свободных в данный момент дополнительных регистров, при этом ссылки на заменяемый регистр во всех последующих командах соответственным образом корректируются. Программист, составляющий программу, имеет дело с именами логических регистров. Число физических регистров аппаратного регистрового файла (АРФ) обычно больше числа логических. «Лишние» регистры АРФ используются в процедуре переименования для временного хранения результатов до момента разрешения конфликтов по данным, после чего значение из регистра временного хранения переписывается на свое «штатное» место. В некоторых процессорах «лишние», регистры в АРФ отсутствуют, а для поддержки переименования предусмотрены специальные структуры, например рассматриваемый *ниже буфер переименования*.

На данном этапе будем считать, что дополнительные физические регистры входят в состав АРФ. Когда выполняется команда, предусматривающая запись результата в какой-то из логических регистров, например в R_n, для временного хране-

ния выделяется один из свободных в данный момент физических регистров АРФ (Rj). Во всех последующих командах, где в качестве логического регистра операнда упоминается Ri, ссылка на него заменяется ссылкой на физический регистр Rj. Таким образом, различные команды, где указан один и тот же логический регистр, могут обращаться к различным физическим регистрам.

Номера логических регистров динамически отображаются на номера физических регистров посредством *таблиц подстановки* (lookup table), которые обновляются после декодирования каждой команды. Очередной результат записывается в новый физический регистр, но значение каждого логического регистра запоминается, благодаря чему легко восстанавливается в случае, если выполнение команды должно быть прервано из-за возникновения исключительной ситуации или неправильного предсказания направления условного перехода.

Переименование регистров может быть реализовано и по-другому - с помощью *буфера переименования*. Проиллюстрируем этот способ, вернувшись к ранее приведенной последовательности из трех команд. Схема содержит аппаратный регистровый файл (АРФ) на *M* регистров и буфер переименования (БП) на N входов (рис. 9.49). Будем считать, что число логических регистров также равно *M,* то есть в качестве временных регистров переименования используются только ячейки БП.

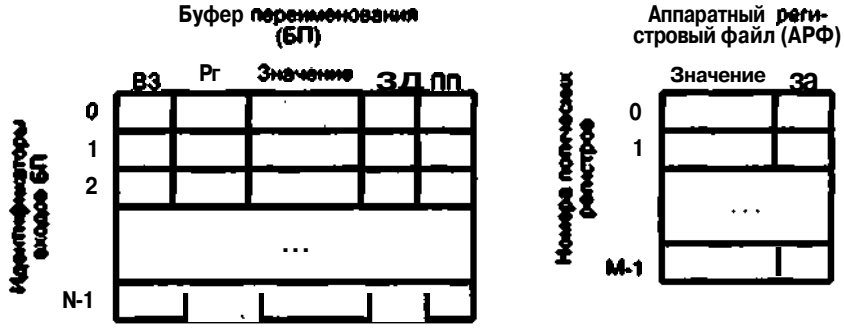


Рис. 9.49. Аппаратный регистровый файл и буфер переименования

Каждому регистру АРФ придан бит «Значение достоверно» (ЗД). Единичное значение ЗД свидетельствует о том, что в регистре содержится корректное значение, которое может быть взято в качестве операнда команды.

Буфер переименования представляет собой ассоциативное запоминающее устройство или набор регистров с ассоциативным доступом. Каждая ячейка или регистр БП идентифицируется своим порядковым номером (0, 1,.,., *N* - 1). Информация, хранящаяся в ячейке или регистре БП (в каждом входе), представляется пятью полями:

- **Вход занят(ВЗ)**. Однобитовое поле, единичное значение которого говорит о том, что этот вход БП недоступен.
- **Номер переименованного регистра (Рг)**. В поле содержится номер логического регистра АРФ, для временной замены которого выделена данная ячейка БП.
- **Значение**. В поле хранится текущее содержимое регистра, указанного в поле Рг.

- **Значение достоверно (ЗД).** Однобитовое поле, единичное значение которого подтверждает достоверность содержимого поля «Значение» (если значение еще не вычислено, то **ЗД** = 0).
- **Последнее переименование (П П).** Если в БП несколько ячеек через поле Рг ссылаются на один и тот же регистр, единица в однобитовом поле ПП будет только у той ячейки, где находится последняя ссылка на данный регистр.

Предположим, что в исходный момент (рис. 9.50, *а)* в буфере переименования заполнены три первых входа, то есть имеем единицы в их поле ВЗ. Текущее состояние БП свидетельствует о том, что в предшествующих командах для записи результатов использовались регистры R4, R0 и Rl, и хотя результаты этих команд уже получены (в поле ЗД записана единица), вычисленные значения еще не переписаны в соответствующие регистры АРФ.

Поле ПП введено из-за того, что регистры могут переименовываться многократно. На рис. 9.50, *6* показан случай последовательного переименования регистра R1 Единица в поле ПП входа 3 указывает, что последнему переименованию регистра R1 соответствует именно данный вход. У всех остальных входов, ссылающихся на «устаревшие» значения R1, в этом поле будет 0. Если очередной команде требуется значение из регистра Rl, то из двух возможных чисел 10 и 15 будет взято 15, то есть значение того входа, где в поле ПП содержится единица.

Всякий раз, когда встречается команда, предполагающая запись результата в регистр, необходимо в буфере переименования выделить для этого регистра один из свободных входов (свободную ячейку) и правильно инициализировать его поля. Обычно буфер заполняется циклически, для чего имеются указатели первого и последнего свободного входов. Когда запрашивается очередной свободный вход, используется указатель «головы», и затем значение этого указателя соответствующим образом изменяется. Когда какой-то из входов освобождается, то корректируется значение указателя «хвоста». После выделения входа для регистра результата производится инициализация его полей: устанавливаются в единицу поля ВД и ПП в поле Рг заносится номер регистра, а в поле ЗД помещается 0, означающий, что поле «Значение» еще не содержит достоверного значения. Рисунок 9.50, *в* иллюстрирует состояние буфера переименования после выделения входа для регистра R2 в команде MUL.

Теперь рассмотрим, каким образом из буфера извлекаются значения операндов (поиск операндов в БП производится, если они отсутствуют в АРФ). В нашем примере это соответствует выборке операндов для команды MUL (значений регистров R0 и R1). Так как рассматриваемый буфер ассоциативный, то для получения значений операндов нужно произвести ассоциативный поиск последних значений регистров R0 и R1, то есть тех входов, где в поле Рг указаны искомые регистры, а в поле ПП содержится 1 (рис. 9.50, *г).*

Если передаваемой далее команде требуется значение регистра, которое еще не вычислено (недостоверно), вместо значения выдается идентификатор (номер) соответствующего входа буфера и ставится пометка, что это не значение, а номер входа. Рисунок 9.50, д показывает такую ситуацию для команды ADD. Первое, что делается при переименовании этой команды, - выделение свободного входа для регистра R3, конкретно - входа *4.* Далее должны быть извлечены значения регистров

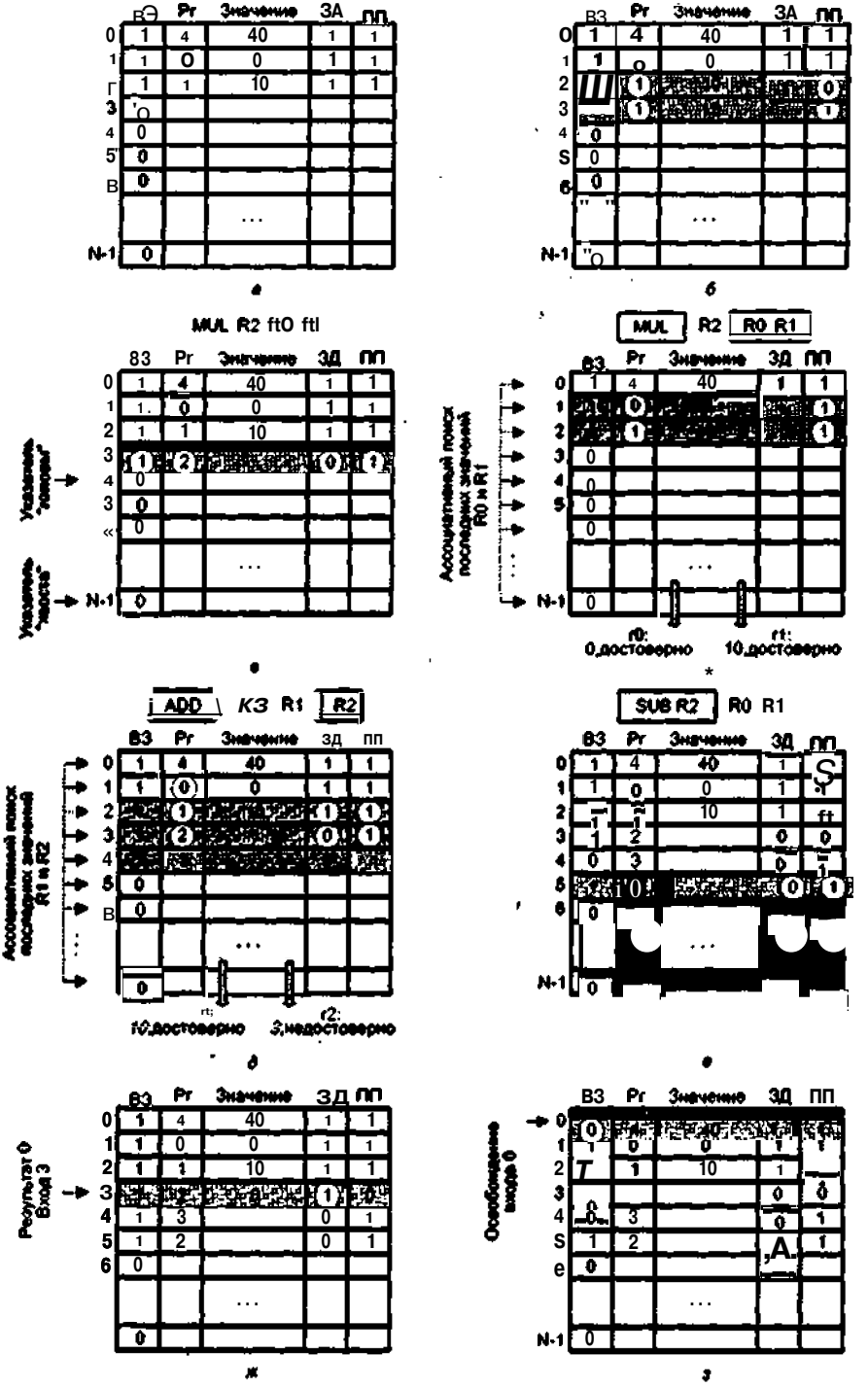


Рис. 9.50. Иллюстрация процессов в буфере переименования

R1 и R2. Поскольку последнее переименование регистра R1 достоверно, выборка содержащегося в регистре значения может быть произведена так, как это было описано выше. Однако значение R2 еще не вычислено, поэтому вместо него в исполнительную часть процессора, где будет выполняться команда ADD, пересылается номер соответствующего-входа буфера переименования (в нашем примере это 3).

Теперь в буфере переименования имеется несколько входов, ссылающихся на один и тот же регистр. Так, на рис. 9.50, *е* показана ситуация, когда команда SUB выдана до завершения команды MUL. В этом случае под регистр R2 выделен еще один вход (вход 5), в котором установлены соответствующие значения в полях ЗД и ПП Одновременно содержимое поля ПП входа 3, где хранилось предыдущее описание регистра R2, изменено на 0. Таким образом, с данного момента все последующие команды, ссылающиеся на регистр R2 как источник операнда, будут переадресовываться на вход 5. Это будет продолжаться до появления новой команды, с регистром результата R2

По завершении команды регистр результата должен быть модифицирован так, чтобы последующие команды могли получить доступ к вычисленному результату. Модификация базируется на идентификаторе входа буфера переименования, выделенного для запрошенного регистра результата. В нашем примере предположим, что завершилась команда MUL и результат 0 должен быть занесен во вход 3 (рисунок *9.50, ж),* В поле ЗП этого входа помещается единица, показывающая, что значение регистра R2 уже доступно.

Последний момент - это освобождение входа буфера переименования (рисунок 9.50, з). Критерий освобождения входа будет рассмотрен позже.

Переупорядочивание команд

После декодирования команд и переименования регистров команды передаются на исполнение. Как уже отмечалось, выдача команд в функциональные блоки может производиться неупорядоченно, по мере готовности. Поскольку порядок выполнения команд может отличаться от предписанного программой, необходимо обеспечить корректность их операндов (частично решается путем переименования регистров) и правильную последовательность занесения результатов в регистры АРФ. Одним из наиболее распространенных приемов решения этих проблем служит *переупорядочивание команд.* В его основе лежат использование *окна команд* — буферной памяти, куда помещаются все команды, прошедшие декодирование, и переименование регистров (последняя операция выполняется только с теми командами, которые записывают свой результат в регистры). Окно команд обеспечивает отсрочку передачи команд на исполнение до момента готовности операндов, а также нужную очередность завершения команд и загрузки их результатов в регистры АРФ. Эта техника известна также под названием *шелвинг* (shelving). Ниже рассматриваются два варианта окна команд — централизованное и распределенное.

Централизованное окно команд. Данное окно реализуется в виде так называемого *табло* (Scoreboard). Техника табло впервые была предложена в 1964 году фирмой Cray и реализована в ЭВМ CDC 6600.

Табло (иногда слово Scoreboard переводят как табельная доска) представляет собой буферное запоминающее устройство, в котором хранится некоторое количество последних извлеченных из памяти и декодированных команд, а также текущая информация о доступности ресурсов, привлекаемых для их исполнения. Функциями табло являются оперативное выявление команд, для исполнения которых уже доступны все необходимые операнды и ресурсы, и выдача таких команд на исполнение в соответствующие функциональные блоки. Табло можно рассматривать как систему предварительной диспетчеризации команд, однако оно осуществляет контроль выполнения команд и после их выдачи.

Все извлеченные из памяти команды сразу же после их декодирования и, если это необходимо, переименования регистров заносятся в табло, причем с соблюдением порядка их следования в программе. Физически табло реализуется на основе ассоциативной памяти. Каждой команде выделяется одна ячейка, состоящая из нескольких полей:

- поля операции, где хранится дешифрированный код операции;
- двух полей операндов, размещающих значения операндов, если они известны, либо информацию о том, откуда эти операнды должны быть получены;
- поля результата, указывающего регистру, куда должен быть помещен результат выполнения данной команды;
- поля битов достоверности.

В табло также хранится текущая информация о доступности устройств обработки (функциональных блоков).

Функционирование табло тесно увязано с работой буфера переименования и может быть описано следующим образом. Каждая команда после декодирования и переименования регистров заносится в очередную свободную ячейку табло. Декодированный код операции помещается в поле операции. Если команда предполагает загрузку результата в регистр, то на этот регистр имеется ссылка в БП и в поле результата заносится номер входа БП, в котором хранится последняя ссылка на данный регистр. Далее делается попытка заполнить поля операндов значениями операндов. Сначала производится поиск нужного значения в аппаратном регистровом файле. Если бит ЗД регистра операнда в АРФ установлен в 0 (значение недостоверно), это означает, что операндом является результат предыдущей операции и дальше следует искать в БП. Выполняется ассоциативный поиск ссылки на регистр в буфере переименования. При удачном исходе (в найденной ячейке БП биты ЗД и ПП установлены в единицу) требуемое значение операнда берется из буфера переименования. В любом варианте при обнаружении достоверного значения операнда поле операнда ячейки табло заполняется найденным значением, а соответствующий этому полю бит достоверности (ЗД) устанавливается в единицу. Если же значение операнда еще не вычислено, то в поле операнда ячейки табло заносится идентификатор входа буфера переименования, где находится последняя ссылка на искомый регистр, при этом бит достоверности такого поля сбрасывается в 0.

Обновление информации о готовности операндов и доступности функциональных устройств выполняется в каждом цикле процессора.

Команда может быть считана из табло и выдана на исполнение лишь после того, как будут занесены значения всех операндов, и лишь при условии, что нужный для исполнения этой команды ФБ свободен. После завершения команды в ФБ производится запись полученного результата (если эта команда предполагает данное действие) в ту ячейку буфера переименования, на которую указывает поле результата. Одновременно производится ассоциативный доступ ко всем хранящимся в табло командам и в тех из них, где в полях операндов указан идентификатор обновленного входа БП, этот идентификатор заменяется занесенным в регистр новым значением, с соответствующей коррекцией битов достоверности. Далее завершенная команда покидает табло. Удаление команды из табло является основанием для перезаписи значения результата данной команды в регистр АРФ и удаления соответствующей записи из буфера переименования.

Отметим, что рассматриваемая технология предполагает схему распределения готовых команд по требуемым для их исполнения функциональным блокам, с одновременной проверкой их доступности. Эта функция на*зва*на *диспетчеризацией*.

В примере, приведенном на рис. 9.51, для команд I1, I2,I3 и I5 известны значения одного из операндов, и они вынуждены ожидать значения второго операнда. Команде I4 известны оба операнда, и при условии доступности ФБ, требуемого для ее исполнения, она вправе быть выдана из окна команд.

Команда	Поле операции	Поле результата	Поле операнде 1	Поле операнда 2	Поле бито» достоверности	
					ЗД1	ЗД2
1	ОП	ИД	ЗН	ИД	ЗН	ИД
2	ОП	ИД	ЗН	ИД	ЗН	ИД
3	ОП	ИД	ИД	ЗН	ИД	ЗН
4	ОП	ИД	ЗН	ЗН	ЗН	ЗН
S	ОП	ИД	ИД	ЗН	ИД	ЗН
			...			

ОП - дешифрированный код операции; ИД - идентификатор регистра;

ЗН - значение операнда; ЗД - значение достоверно

Рис. 9. 51 . Содержимое табло

В каждом такте работы процессора готовыми к выдаче могут оказаться сразу несколько команд, и все готовые команды должны быть направлены в соответствующие функциональные блоки. Если имеется несколько однотипных блоков обработки, то в процессоре должна быть предусмотрена логика выбора одного из них.

После выдачи команды из табло ее позиция освобождается и может быть использована для загрузки новой команды. Вместе с тем необходимо сохранить заданную программой последовательность команд. Эту задачу решают одним из двух методов. В первом из них, именуемом как *стек диспетчеризации,* после выдачи команд и освобождения позиций в окне последующие команды сдвигаются вниз, заполняя вновь доступные позиции и освобождая верхнюю часть табло. Новые команды всегда загружаются в верхнюю часть табло. В случае второго метода, с так *наз*ыв*аемым блоком обновления регистров,* табло функционирует так же, как оче-

редь типа FIFO, но производится общий сдвиг вниз, включая и освободившиеся позиции. Это упрощает логику работы централизованного окна.

Распределенное окно команд. В варианте распределенного окна команд на входе каждого функционального блока размещается буфер декодированных команд, называемый *накопителем команд* или *схемой резервирования* (reservation station). Метод резервирования был разработан Р. Л. Томасуло в 1967 году и впервые воплощен в вычислительной системе IBM 360/91. После выборки и декодирования команды распределяются по схемам резервирования тех ФБ, где команда будет исполняться. В буфере команда запоминается и по готовности выдается в связанный с данным пунктом функциональный блок. Логика работы каждого накопителя аналогична централизованному окну команд. Выдача происходит только после того, как команда получит все необходимые операнды, и при условии, что ФБ свободен. При обновлении содержимого буфера переименования файла производится доступ ко всем накопителям команд, и в них идентификаторы обновленных входов заменяются хранящимися в этих входах значениями операндов.

Отметим одну особенность рассматриваемой схемы: не требуется, чтобы операнд был обязательно занесен в отведенный для него регистр - он может быть ускоренно передан прямо в накопитель команд для немедленного использования или буферизирован там для последующего использования.

Число независимых команд, которые могут выполняться одновременно, варьируется от программы к программе, а также в пределах каждой программы. В среднем число таких команд равно 1 - 3, временами возрастая до 5-6. Механизм резервирования ориентирован на одновременную выдачу нескольких команд, что, как правило, легче реализовать с распределенным, а не централизованным окном команд, поскольку темп загрузки распределенных буферов обычно меньше, чем потенциальный темп выдачи команд. Пропускная способность линии связи между централизованным окном команд и функциональными блоками должна быть выше, чем в случае распределенного окна. Однако для централизованного окна характерно более эффективное задействование емкости буфера.

Емкость накопителя команд в каждом функциональном блоке зависит от ожидаемого числа команд для этого блока. Типичный накопитель рассчитан на 1-3 команды. Если в одной из них одновременно готовы несколько команд, выдача их в ФБ производится в порядке занесения этих команд в накопитель.

Для более детального пояснения процессов, происходящих в технологии накопителей, рассмотрим следующий пример. На рис. 9.52 показана схема передачи декодированных команд с переименованными регистрами в накопитель команд. Предполагается, что в рамках одного цикла в накопитель могут быть выданы до двух команд. Буфер переименования представлен регистровым файлом (РгФ). Из РгФ позволено одновременно выбрать по два операнда (R_{S1}, R_{S2}) для каждой из двух команд. Каждый регистр РгФ имеет дополнительный бит достоверности *(V),* единичное состояние которого свидетельствует о корректности содержимого регистра. Доступные операнды (O_{S1},0_{S2}) переписываются в соответствующие поля ячеек накопителя (O_{S1}/I_{S1}, O_{S2}/I_{S2}), при этом биты достоверности этих полей (V_{S1}, V_{S2}) устанавливаются в единицу. Если значение операнда в РгФ недоступно, то в поля (O_{S1}/I_{S1}, O_{S2}/I_{S2})заносится порядковый номер того регистра, откуда операнд должен быть

получен (обозначаются как I_{S1} и I_{S2}), а в соответствующий бит (V_{S1} или V_{S2}) записывается 0. Если выдаваемая в регистр команда предполагает запись результата в регистр R_O, то бит достоверности этого регистра в РгФ сбрасывается, тем самым запрещается последующим командам использовать содержимое данного регистра.

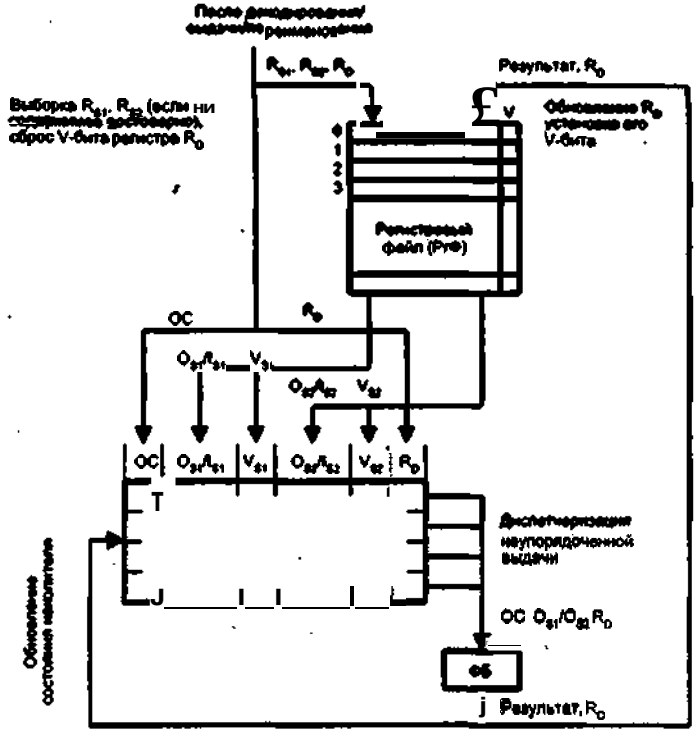


Рис. 9.52. Общая схема шелвинга к примеру

После передачи команд в накопитель там производится проверка на наличие команд, для исполнения которых есть все необходимые операнды. Поиск таких команд выполняется путем анализа битов V_{S1} и V_{S2}Если у команды оба бита в единичном состоянии, то она готова к выдаче в ФБ. При наличии в накопителе только одной команды она сразу выдается в ФБ. Если готовых команд несколько, из них в ФБ пересылается наиболее «старая», то есть поступившая в накопитель первой. После завершения команды ее результат совместно с идентификатором регистра результата (R_o) выдается в РгФ и в накопитель для обновления их содержимого. В ходе обновления РгФ вычисленное значение заносится в RD, а бит достоверности регистра результата устанавливается в единицу. С этого момента значение R_O доступно в качестве операнда для последующих команд. В накопителе производятся поиск идентификатора R_O, в полях (O_{S1}/I_{S1}, O_{S2}/I_{S2}) всех команд и их замена на вычисленное значение. Одновременно состояние бита V_{S1} (V_{Si}) изменяется на единицу. Далее выполняются очередной поиск готовых к исполнению команд и их выдача в ФБ.

Теперь рассмотрим технику переупорядочивания команд с использованием накопителей на примере следующей последовательности команд:

MUL R3, R1, R2{R3 ← Rl x R2}
ADD **R5,** R2, **R3{R5 ← R2 + R3}**
ADD R6, **R3, R4 {R6 ←** R3 + **R4}.**

После декодирования и переименования регистров команда MUL в цикле *г* выдается в накопитель (рис. 9.53, *а).* Одновременно с этим из РгФ выбираются операнды этой команды (Rl и R2). Поскольку биты достоверности регистров операндов (V-биты) показывают, что значения операндов доступны (R1 - 10, R2 - 20), то эти значения будут переданы в накопитель, а биты наличия операндов в накопителе (V_{S1} и V_a) будут установлены в единицу. Бит достоверности регистра назначения R3 в РгФ сбрасывается с тем, чтобы не допустить доступ к этому регистру последующих команд до тех пор, пока в него не будет помещен результат операции MUL.

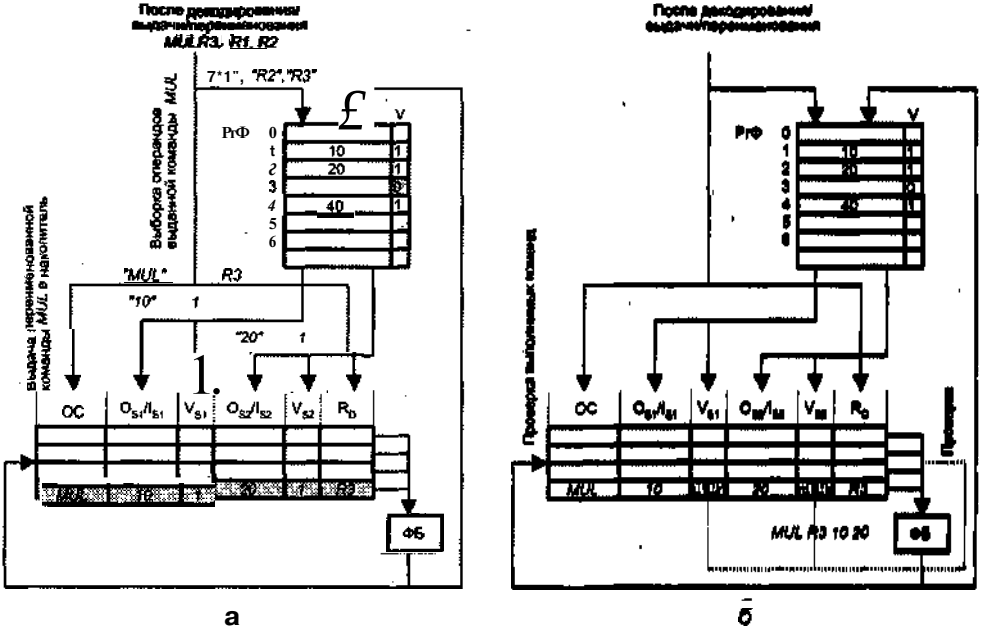


Рис. 9.53. К примеру техники переупорядочивания команд с использованием накопителей: а — выдача команды MUL в накопитель в цикле i и выборка соответствующих операндов; *б* — поиск выполнимых команд и диспетчеризация команды MUL в цикле i +1

Вследующем(*' + 1)-м цикле происходит передача команды из накопителя (этот этап принято называть диспетчеризацией) в функциональный блок (ФБ) и выдача в накопитель двух очередных команд ADD. Сначала делается проверка битов наличия операндов (V_{S1} и V_{S2}) у всех находящихся в накопителе команд (в нашем примере здесь только одна команда MUL), Поскольку оба операнда доступны (V_{S1} - 1 и V_{S2} - 1), команда MUL пересылается в ФБ на исполнение. Это иллюстрирует рис. 9.53,*б.*

Общая форма параллелизма на уровне программ проистекает из разбиения программируемых данных на подмножества. Это разделение называют *декомпозицией области* (domain decomposition), а параллелизм, возникающий при этом, носит название *параллелизма данных.* Подмножества данных назначаются разным вычислительным процессам, и называется этот процесс *распределением данных* (data distribution). Процессоры выделяются определенным процессам либо по инициативе программы, либо в процессе работы операционной системой. На каждом процессоре может выполняться более чем один процесс.

Параллелизм уровня команд

Параллелизм на уровне команд имеет место, когда обработка нескольких команд или выполнение различных этапов одной и той же команды может перекрываться во времени. Разработчики вычислительной техники издавна прибегали к методам, известным под общим названием «совмещения операций*-, при котором аппаратура ВМ в любой момент времени выполняет одновременно более одной операции. Этот общий принцип включает в себя два понятия: *параллелизм* и *конвейеризацию.* Хотя у них много общего и их зачастую трудно различать на практике, термины эти отражают два принципиально различных подхода.

В первом варианте совмещение операций достигается за счет того, что в составе вычислительной системы отдельные устройства присутствуют в нескольких копиях Так, в состав процессора может входить несколько АЛУ, и высокая производительность обеспечивается за счет одновременной работы всех этих АЛУ. Второй подход был описан ранее.

Метрики параллельных вычислений

В силу особенностей параллельных вычислений для оценки их эффективности используют специфическую систему метрик. Наиболее распространенные из таких метрик рассматриваются ниже.

Профиль параллелизма программы

Число процессоров многопроцессорной системы, параллельно участвующих в выполнении программы в каждый момент времени *t,* определяют понятием *степень параллелизма D(t)* (DOP, Degree Of Parallelism). Графическое представление параметра *D(t)* как функции времени называют *профилем параллелизма программы.* Изменения в уровне загрузки процессоров за время наблюдения зависят от многих факторов (алгоритма, доступных ресурсов, степени оптимизации, обеспечиваемой компилятором и т. д.). Типичный профиль параллелизма для алгоритма декомпозиции (divide-and-conquer algorithm) показан на рис. 10.1.

В дальнейшем будем исходить из следующих предположений: система состоит из л гомогенных процессоров; максимальный параллелизм в профиле равен m и, в идеальном случае, $n >> m$. Производительность Δ одиночного процессора системы выражается в единицах скорости вычислений (количество операций в единицу времени) и не учитывает издержек, связанных с обращением к памяти и пересылкой данных. Если за наблюдаемый период загружены і процессоров, то $D - i$.

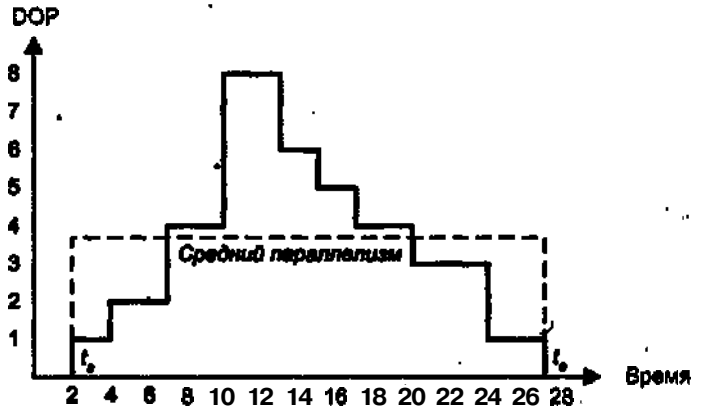


Рис. 10.1. Профиль параллелизма

Общий объем вычислительной работы W (команд или вычислений), выполненной начиная со стартового момента t_s до момента завершения t_c, пропорционален площади под кривой профиля параллелизма:

$$W = \Delta \int_{t_s}^{t_e} D(t)dt.$$

Интеграл часто заменяют дискретным эквивалентом:

$$W = \Delta \sum_{i=1}^{m} i \times t_i,$$

где t_i — общее время, в течение которого D — i, а $\sum_{i=1}^{m} t_i = t_e - t_s$ — **общее** затраченное время.

Средний параллелизм A опред $= \frac{1}{t_e - t_s} \int_{t_s}^{t_e} D(t)dt$

В дискретной форме это можн $A = \frac{\sum_{i=1}^{m} i \times t_i}{\sum_{i=1}^{m} t_i}$ к:

Профиль параллелизма на рисунке за время наблюдения (t_s, t) возрастает от 1 до пикового значения $m = 8$, азатем спадает до 0. Средний параллелизм A = (1x5 + + 2x3 + 3x4 + 4x6 + 5x2 + 6x2 + 8x3) / (5+ 3 + 4 + 6 + 2 + 2 + 3) - 93/25- = 3,72. Фактически общая рабочая нагрузка и A представляют собой нижнюю границу асимптотического ускорения.

Будем говорить, что параллельная программа выполняется в режиме i, если для ее исполнения задействованы i процессоров. Время, на продолжении которого система работает в режиме i, обозначим через t_i а объем работы, выполненной в ре-

жиме i, как $W_i = i\Delta t_i$. Тогда, при наличии и процессоров время исполнения $T(n)$ для двух экстремальных случаев: $n = 1 (T(1))$ и $n = \infty (T(\infty))$ можно записать в виде:

$$T(1) = \sum_{i=1}^{m} t_i(1) = \sum_{i=1}^{m} \frac{W_i}{\Delta} \text{ и } T(\infty) = \sum_{i=1}^{m} t_i(\infty) = \sum_{i=1}^{m} \frac{W_i}{i\Delta}.$$

Асимптотическое повышение быстродействия 5_ определяется как отношение $T(1)$ к $T(n)$:

$$S_\infty = \frac{T(1)}{T(\infty)} = \frac{\sum_{i=1}^{m} W_i}{\sum_{i=1}^{m} \frac{W_i}{i}}.$$

Сравнивая это выражение и предыдущие уравнения, можно констатировать, что в идеальном варианте $S_\infty = A$. В общем случае нужно учитывать коммуникационные задержки и системные издержки. Отметим, что как S_∞, так и A были определены в предположении, что $n >> m$.

В прикладных программах имеется широкий диапазон потенциального параллелизма М. Кумар в своей статье [146] приводил данные, что в вычислительно интенсивных программах в каждом цикле параллельно могут выполняться от 500 до 3500 арифметических операций, если для этого имеется существующая вычислительная среда. Однако даже правильно спроектированный суперскалярный процессор способен поддерживать выполнение от 2 до 5,8 команды за цикл. Эти цифры дают пессимистическую картину возможного параллелизма.

Ускорение, эффективность, загрузка и качество

Рассмотрим параллельное выполнение программы со следующими характеристи-

- 0(n) - общее число операций (команд), выполненных на «-процессорной системе;
- *T(n)* - время выполнения *O(n)* операций на n-процессорной системе в виде числа квантов времени.

В общем случае $T(n) < O(n)$, если в единицу времени n процессорами выполняется более чем одна команда, где $n > 2$. Примем, что в однопроцессорной системе $T(1) = o(1)$.

Ускорение (speedup), или точнее, среднее ускорение за счет параллельного выполнения программы - это отношение времени, требуемого для выполнения наилучшего из последовательных алгоритмов на одном процессоре, и времени параллельного вычисления на n процессорах. Без учета коммуникационных издержек ускорение $S(n)$ определяется как

$$S(n) = \frac{T(1)}{T(n)}.$$

Как правило, ускорение удовлетворяет условию $S(n) <= n$.

Эффективность (efficiency) n-процессорной системы - это ускорение на один процессор, определяемое выражением

$$\text{ад}.\text{ад}\ldots\text{ха})\quad .$$
$$n \qquad n \times T(n)$$

Эффективность обычно отвечает условию 1/и < =*E(n)* < =*n*. Для более реалистичного описания производительности параллельных вычислений необходимо промоделировать ситуацию, когда параллельный алгоритм может потребовать больше операций, чем его последовательный аналог.

Довольно часто организация вычислений на *n* процессорах связана с существенными издержками. Поэтому имеет смысл ввести понятие *избыточности* (redundancy) в виде

$$R(n) = \frac{O(n)}{O(1)} = \frac{1}{\text{ад}} = 1.$$

Это отношение отражает степень соответствия между программным и аппаратным параллелизмом. Очевидно, что 1 < =*R(n)* < =*n*.

Определим еще одно понятие, *коэффициент полезного использования* или *ут*илизации (utilization), как

$$\text{ед-едхвд} = \frac{\text{ff}}{n \times T(n)}$$

Тогда можно утверждать, что

$$\frac{1}{n} \le E(n) \le U(n) \le 1 \quad \text{и} \quad 1 < R(n) < \frac{1}{E(n)} \le n.$$

Рассмотрим пример. Пусть наилучший из известных последовательных алгоритмов занимает 8 с, а параллельный алгоритм занимает на пяти процессорах 2 с Тогда:

- *S(n) =8/2 = A;*
- *E*(n)=4/5 = 0,8;
- *R(n)*= 1/0,8- 1=0,25.

Собственное ускорение определяется путем реализации параллельного алгоритма на одном процессоре.

Если ускорение, достигнутое на *n* процессорах, равно и, то говорят, что алгоритм *показывает линейное ускорение*.

В исключительных ситуациях ускорение S(n) может быть больше, чем и. В этих случаях иногда применяют термин *суперлинейное ускорение*. Данное явление имеет шансы на возникновение в двух следующих случаях:

- Последовательная программа может выполняться в очень маленькой памяти, вызывая свопинги (подкачки), или, что менее очевидно, может приводить к большему числу кэш-промахов, чем параллельная версия, где обычно каждая параллельная часть кода выполняется с использованием много меньшего набора данных.
- Другая причина повышенного ускорения иллюстрируется примером. Пусть нам нужно выполнить логическую операцию A_1 v A_2 где как A_1 так и A_2 имеют значение «Истина» с вероятностью 50%, причем среднее время вычисленияA_i, обо-

значенное как $T(A_i)$, существенно различается в зависимости от того, является ли результат истинным или ложным.

Пусть $T(A_i) = \begin{cases} 1 \text{ с} & \text{для } A_i = 1; \\ \text{для } A_i = 0 & \end{cases}$

$T(A_i) = \begin{cases} 1 \text{ с} & \text{для } A_i = 1; \\ 100 \text{ с} & \text{для } A_i = 0 \end{cases}$

AT	A*	Последовательный	Параллельный
T	T	1с + 0	1с
T	F	1с + 0	1с
F	T	100с + 1с	1с
F	F	100 с + 100 с	100 с
Г		$\approx \frac{303}{4}$ с ≈ 76 с	$\approx \frac{103}{4}$ с ≈ 26 с

Таким образом, параллельные вычисления на двух процессорах ведут к среднему ускорению:

$$S(2) = \frac{T(1)}{\Gamma<2)} = \frac{303}{103} - 3.$$

Отметим, что суперлинейное ускорение вызвано жесткостью последовательной обработки, так как после вычисления еще нужно проверить A_2.

К факторам, ограничивающим ускорение, следует отнести:

- **Программные издержки.** Даже если последовательные и параллельные алгоритмы выполняют одни и те же вычисления, параллельным алгоритмам присущи добавочные программные издержки — дополнительные индексные вычисления, неизбежно возникающие из-за декомпозиции данных и распределения их по процессорам; различные виды учетных операций, требуемые в параллельных алгоритмах, но отсутствующие в алгоритмах последовательных.
- **Издержки из-за дисбаланса загрузки процессоров.** Между точками синхронизации каждый из процессоров должен быть загружен одинаковым объемом работы, иначе часть процессоров будет ожидать, пока остальные завершат свои операции. Эта ситуация *известна как дисбаланс загрузки*. Таким образом, ускорение ограничивается наиболее медленным из процессоров.
- **Коммуникационные издержки.** Если принять, что обмен информацией и вычисления могут перекрываться, то любые коммуникации между процессорами снижают ускорение. В плане коммуникационных затрат важен уровень гранулярности, определяющий объем вычислительной работы, выполняемой между коммуникационными фазами алгоритма. Для уменьшения коммуникационных издержек выгоднее, чтобы вычислительные гранулы были достаточно крупными и доля коммуникаций была меньше.

Еще одним показателем параллельных вычислений служит *качество* параллельного выполнения программ - характеристика, объединяющая ускорение, эффективность и избыточность. Качество определяется следующим образом:

$$Q(n) = \frac{S(n) \times E(n)}{R(n)} = \frac{T^3(1)}{n \times T^2(n) \times O(n)}.$$

Поскольку как эффективность, так и величина, обратная избыточности, представляют собой дроби, то Q(n) < =*S(n)*. Поскольку *E(n)* - это всегда дробь, а *R(n)* - число между 1 и и, качество *Q(n)* при любых условиях ограничено сверху величиной ускорения S(n).

Закон Амдала

Приобретая для решения своей задачи параллельную вычислительную систему, пользователь рассчитывает на значительное повышение скорости вычислений за счет распределения вычислительной нагрузки по множеству параллельно работающих процессоров. В идеальном случае система из и процессоров могла бы ускорить вычисления в *n* раз, В реальности достичь такого показателя по ряду причин не удается. Главная из этих причин заключается в невозможности полного распараллеливания ни одной из задач. Как правило, в каждой программе имеется фрагмент кода, который принципиально должен выполняться последовательно и только одним из процессоров. Это может быть часть программы, отвечающая за запуск задачи и распределение распараллеленного кода по процессорам, либо фрагмент программы, обеспечивающий операции ввода/вывода. Можно привести и другие примеры, но главное состоит в том, что о полном распараллеливании задачи говорить не приходится. Известные проблемы возникают и с той частью задачи, которая поддается распараллеливанию. Здесь идеальным был бы вариант, когда параллельные ветви программы постоянно загружали бы все процессоры системы, причем так, чтобы нагрузка на каждый процессор была одинакова. К сожалению, оба этих условия на практике трудно реализуемы. Таким образом, ориентируясь на параллельную ВС, необходимо четко сознавать, что добиться прямо пропорционального числу процессоров увеличения производительности не удастся, и, естественно, встает вопрос о том, на какое реальное ускорение можно рассчитывать. Ответ на этот вопрос в какой-то мере дает закон Амдала.

Джин Амдал (Gene Amdahl) - один из разработчиков всемирно известной Системы IBM 360, в своей работе [48], опубликованной в 1967 году, предложил формулу, отражающую зависимость ускорения вычислений, достигаемого на многопроцессорной ВС, от числа процессоров и соотношения между последовательной и параллельной частями программы. Показателем сокращения времени вычислений служит такая метрика, как *"ускорение"*. Напомним, что ускорение 5 - это отношение времени T_s, затрачиваемого на проведение вычислений на однопроцессорной ВС (в варианте наилучшего последовательного алгоритма), ко времени Т, решения той же задачи на параллельной системе (при использовании наилучшего параллельного алгоритма):

$$S = \frac{T_s}{m_p}.$$

Оговорки относительно алгоритмов решения задачи сделаны, чтобы подчеркнуть тот факт, что для последовательного и параллельного решения лучшими мо-

гут оказаться разные реализации, а при оценке ускорения необходимо исходить именно из наилучших алгоритмов.

Проблема рассматривалась Амдалом в следующей постановке (рис. 10.2). Прежде всего, объем решаемой задачи с изменением числа процессоров, участвующих в ее решении-, остается неизменным. Программный код решаемой задачи состоит из двух частей: последовательной и распараллеливаемой. Обозначим долю операций, которые должны выполняться последовательно одним из процессоров, через f, где 0 <=f<= 1 (здесь доля понимается не по числу строк кода, а по числу реально выполняемых операций). Отсюда доля, приходящаяся на распараллеливаемую часть программы, составит 1 -f. Крайние случаи в значениях/соответствуют полностью параллельным (f= 0) и полностью последовательным (f- 1) программам. Распараллеливаемая часть программы равномерно распределяется по всем процессорам. ,

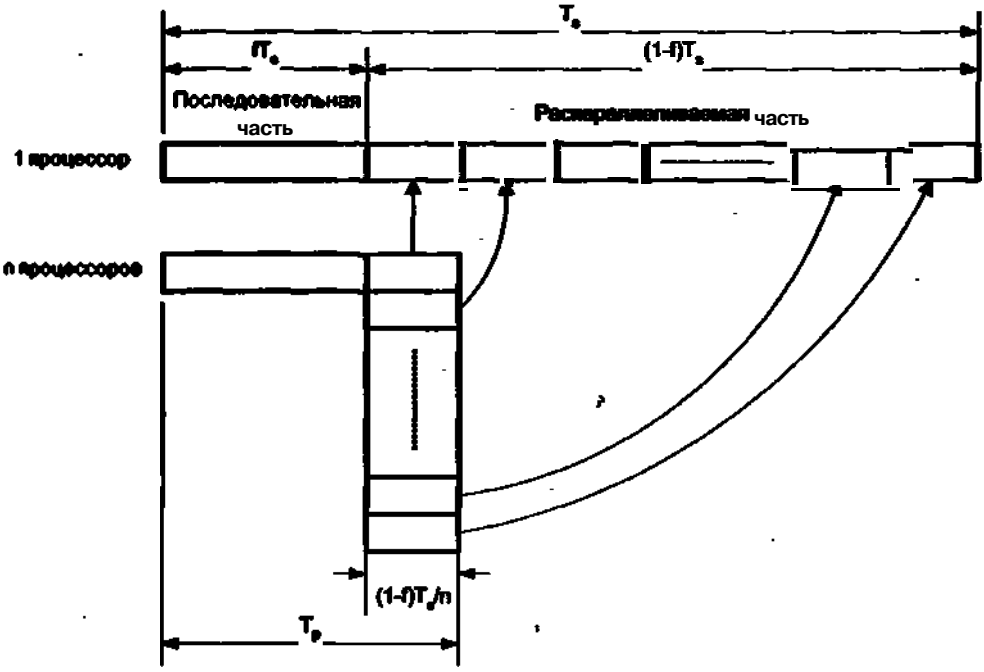


Рис. 10.2. К постановке задачи в законе Амдала

С учетом приведенной ф

$$T_p = f \times T_s + \frac{(1-f) \times T_s}{n}.$$

В результате получаем формулу Амдала, выражающую ускорение, которое может быть достигнуто на системе из *n* процессоров:

$$S = \frac{T_s}{T_p} = \frac{n}{1 + (n-1) \times f}.$$

Формула выражает простую и обладающую большой общностью зависимость. Характер зависимости ускорения от числа процессоров и доли последовательной. части программы показан на рис. 10.3.

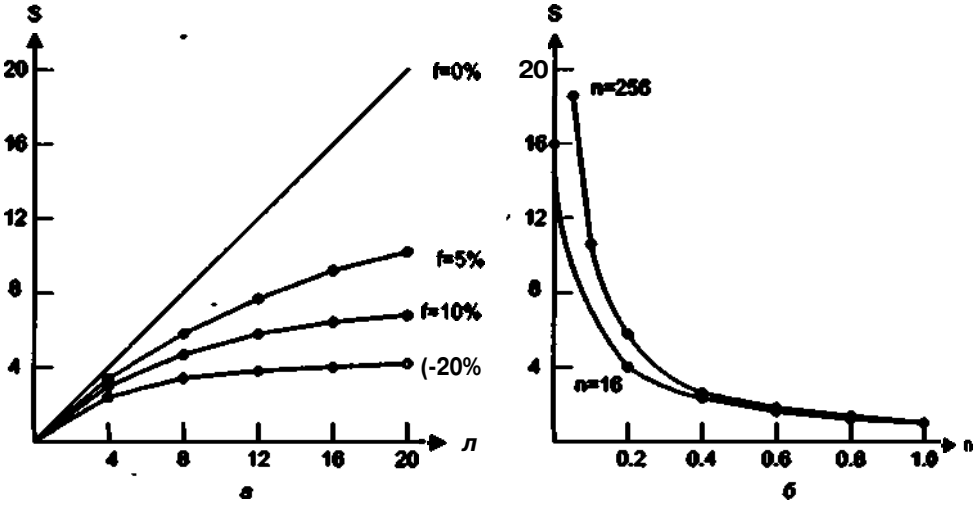


Рис. 10.3. Графики зависимости ускорения от: *а* — доли последовательных вычислений; *б* — числа процессоров

Если устремить число процессоров к бесконечности, то в пределе получаем:

$$\lim_{n\to\infty} S = \frac{1}{f}.$$

Это означает, что если в программе 10% последовательных операций (то есть /-0,1), то, сколько бы процессоров ни использовалось, убыстрения работы программы более чем в десять раз никак ни получить, да и то, 10 — это теоретическая верхняя оценка самого лучшего случая, когда никаких других отрицательных факторов нет. Следует отметить, что распараллеливание ведет к определенным издержкам, которых нет при последовательном выполнении программы. В качестве примера таких издержек можно упомянуть дополнительные операции, связанные с распределением программ по процессорам, обмен информацией между процессорами и т. д.

Закон Густафсона

Известную долю оптимизма в оценку, даваемую законом Амдала, вносят исследования, проведенные уже упоминавшимся Джоном Густафсоном из NASA Ames Research [115]. Решая на вычислительной системе из 1024 процессоров три больших задачи, для которых доля последовательного кода/лежала в пределах от 0,4 до 0,8%, он получил значения ускорения по сравнению с однопроцессорным вариантом, равные соответственно 1021,1020 и 1016. Согласно закону Амдала для данного числа процессоров и диапазона f, ускорение не должно было превысить вели-

чины порядка 201. Пытаясь объяснить это явление, Густафсон пришел к выводу, что причина кроется в ИСХОДНОЙ предпосылке, лежащей в основе закона Амдала: увеличение числа процессоров не сопровождается увеличением объема решаемой задачи. Реальное же поведение пользователей существенно отличается от такого представления. Обычно, получая в свое распоряжение более мощную систему, пользователь не стремится сократить время вычислений, а, сохраняя его практически неизменным, старается пропорционально мощности ВС увеличить объем решаемой задачи. И тут оказывается, что наращивание общего объема программы касается главным образом распараллеливаемой части программы. Это ведет к сокрашению значения f. Примером может служить решение дифференциального уравнения в частных производных. Если доля последовательного кода составляет 10% для 1000 узловых точек, то для 100 000 точек доля последовательного кода снизится до 0,1%. Сказанное иллюстрирует рис. 10.4, который отражает тот факт, что, оставаясь практически неизменной, последовательная часть в общем объеме увеличенной программы имеет уже меньший удельный вес.

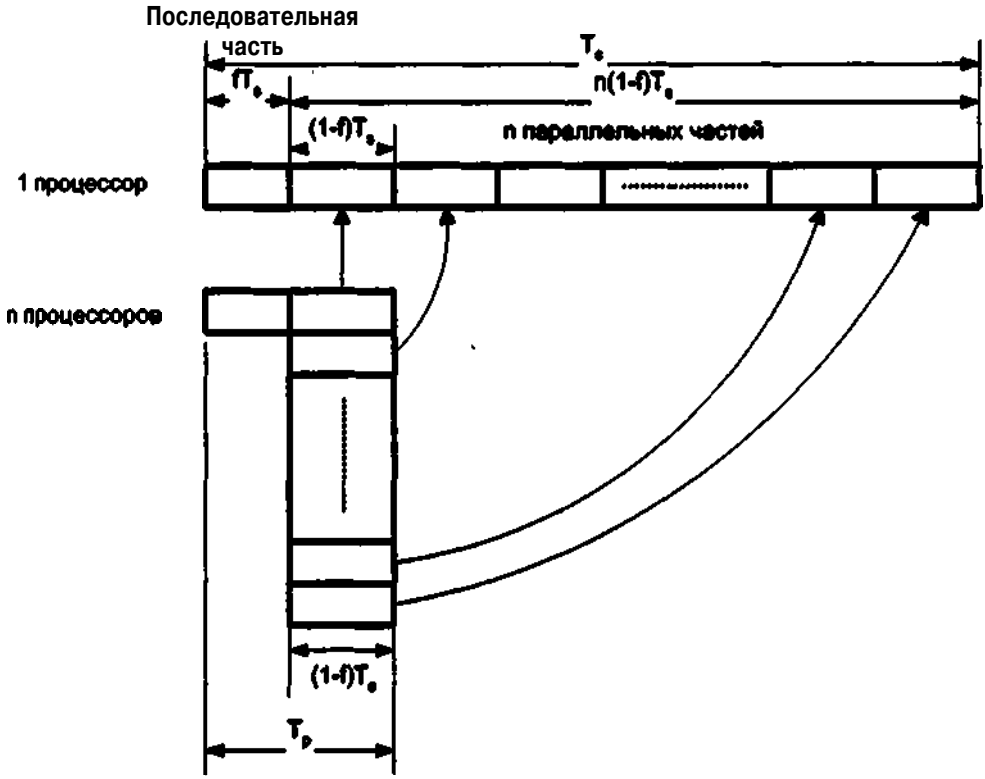


Рис. 10.4. К постановке задачи в законе Густафсона

Было отмечено, что в первом приближении объем работы, которая может быть произведена параллельно, возрастает линейно с ростом числа процессоров в системе. Для того чтобы оценить возможность ускорения вычислений, когда объем

последних увеличивается с ростом количества процессоров в системе (при постоянстве общего времени вычислений), Густафсон рекомендует использовать выражение, предложенное Е. Барсисом (E. Barsis):

$$S = \frac{T_s}{T_p} = \frac{f \times T_s + n \times (1-f) \times T_s}{f \times T_s + (1-f) \times T_s} = n + (1-n) \times f.$$

Данное выражение известно *как закон масштабируемого ускорения* или *закон Густафсона* (иногда его называют также законом Густафсона-Барсиса), В заключение отметим, что закон Густафсона не противоречит закону Амдала. Различие состоит лишь в форме утилизации дополнительной мощности ВС, возникающей при увеличении числа процессоров.

Классификация параллельных вычислительных систем

Даже краткое перечисление типов современных параллельных вычислительных систем (ВС) дает понять, что для ориентирования в этом многообразии необходима четкая система классификации. От ответа на главный вопрос — что заложить в основу классификации - зависит, насколько конкретная система классификации помогает разобраться с тем, что представляет собой архитектура ВС и насколько успешно данная архитектура позволяет решать определенный круг задач. Попытки систематизировать все множество архитектур параллельных вычислительных систем предпринимались достаточно давно и длятся по сей день, но к однозначным выводам пока не привели. Исчерпывающий обзор существующих систем классификации ВС приведен в [5].

Классификация Флинна

Среди всех рассматриваемых систем классификации ВС наибольшее признание получила классификация, предложенная в 1966 году М. Флинном [99, 100]. В ее основу положено понятие потока, под которым понимается последовательность элементов, команд или данных, обрабатываемая процессором. В зависимости от количества потоков команд и потоков данных Флинн выделяет четыре класса архитектур: SISD, MISD, SIMD, MIMD.

SISD

SISD (Single Instruction Stream/Single Data Stream) - одиночный поток команд и одиночный поток данных (рис. 10.5, *а)*. Представителями этого класса являются, прежде всего, классические фон-неймановские ВМ, где имеется только один поток команд, команды обрабатываются последовательно и каждая команда инициирует одну операцию с одним потоком данных. То, что для увеличения скорости обработки команд и скорости выполнения арифметических операций может применяться конвейерная обработка, не имеет значения, поэтому в класс SISD одновременно попадают как ВМ CDC 6600 со скалярными функциональными устройствами, так и CDC 7600 с конвейерными. Некоторые специалисты считают, что к SISD-системам можно причислить и векторно-конвейерные ВС, если рассматривать вектор как неделимый элемент данных для соответствующей команды.

MISD

MISD (Multiple Instruction Stream/Single Data Stream) - множественный поток команд и одиночный поток данных (рис. 10.5, б). Из определения следует, что в архитектуре ВС присутствует множество процессоров, обрабатывающих один и тот же поток данных. Примером могла бы служить ВС, на процессоры которой подается искаженный сигнал, а каждый из процессоров обрабатывает этот сигнал с помощью своего алгоритма фильтрации. Тем не менее ни Флинн, ни другие специалисты в области архитектуры компьютеров до сих пор не сумели представить убедительный пример реально существующей вычислительной системы, построенной на данном принципе. Ряд исследователей [117,121,213] относят к данному классу конвейерные системы, однако это не нашло окончательного признания. Отсюда принято считать, что пока данный класс пуст. Наличие пустого класса не следует считать недостатком классификации Флинна. Такие классы, по мнению некоторых исследователей [84,192], могут стать чрезвычайно полезными для разработки принципиально новых концепций в теории и практике построения вычислительных систем.

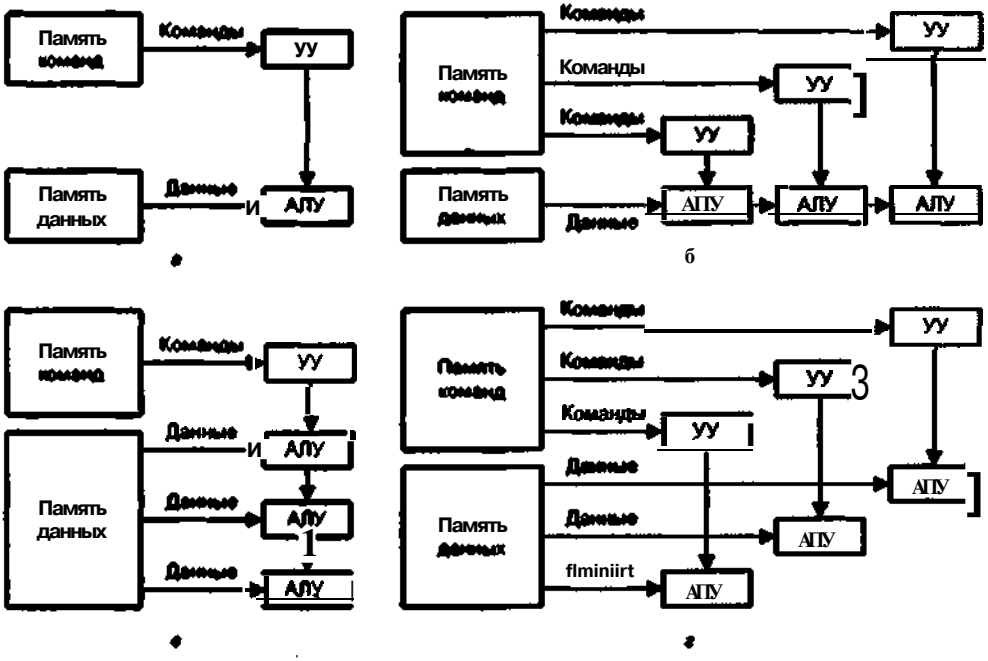


Рис. 1 0.5. Архитектура вычислительных систем по Флинну: *а* - SISD; *б* - MISD; в —SIMD;г —MIMD

SIMD

SIMD (Single Instruction Stream/Multiple Data Stream) - одиночный поток команд и множественный поток данных (рис. 10.5, *в)*. ВМ данной архитектуры позволяют выполнять одну арифметическую операцию сразу над многими данными — элементами вектора. Бесспорными представителями класса SIMD считаются матрицы

процессоров, где единое управляющее устройство контролирует множество процессорных элементов. Все процессорные элементы получают от устройства управления одинаковую команду и выполняют ее над своими локальными данными. В принципе в этот класс можно включить и векторно-конвейерные ВС, если каждый элемент вектора рассматривать как отдельный элемент потока данных.

MIMD

MIMD (Multiple Instruction Stream/Multiple Data Stream) - множественный поток команд и множественный поток данных (рис. 10.5, г). Класс предполагает наличие в вычислительной системе множества устройств обработки команд, объединенных в единый комплекс и работающих каждое со своим потоком команд и данных. Класс MIMD чрезвычайно широк, поскольку включает в себя всевозможные мультипроцессорные системы. Кроме того, приобщение к классу MIMD зависит от трактовки. Так, ранее упоминавшиеся векторно-конвейерные ВС можно вполне отнести и к классу MIMD, если конвейерную обработку рассматривать как выполнение множества команд (операций ступеней конвейера) над множественным скалярным потоком.

Схема классификации Флинна вплоть до настоящего времени является наиболее распространенной при первоначальной оценке той или иной ВС, поскольку позволяет сразу оценить базовый принцип работы системы, чего часто бывает достаточно. Однако у классификации Флинна имеются и очевидные недостатки, например неспособность однозначно отнести некоторые архитектуры к тому или иному классу. Другая слабость - это чрезмерная насыщенность класса MIMD. Все это породило множественные попытки либо модифицировать классификацию Флинна, либо предложить иную систему классификации.

Контрольные вопросы

1. Сравните схемы классификации параллелизма по уровню и гранулярности. Каковы, на ваш взгляд, достоинства, недостатки и области применения этих схем классификации?
2. Для заданной программы и конфигурации параллельной вычислительной системы рассчитайте значения метрик параллельных вычислений.
3. Поясните суть закона Амдала, приведите примеры, поясняющие его ограничения.
4. Какую проблему закона Амдала решает закон Густафсона? Как он это делает? Сформулируйте области применения этих двух законов.
5. Укажите достоинства и недостатки схемы классификации Флинна.

Глава 11
Организация памяти вычислительных систем

В вычислительных системах, объединяющих множество параллельно работающих процессоров или машин, задача правильной организации памяти является одной из важнейших. Различие между быстродействием процессора и памяти всегда было камнем преткновения в однопроцессорных ВМ. Многопроцессорность ВС приводит еще к одной проблеме - проблеме одновременного доступа к памяти со стороны нескольких процессоров.

В зависимости от того, каким образом организована память многопроцессорных (многомашинных) систем, различают вычислительные системы с общей памятью (shared memory) и ВС с распределенной памятью (distributed memory). В *системах с общей памятью* (ее часто называют также совместно используемой или разделяемой памятью) память ВС рассматривается как общий ресурс, и каждый из процессоров имеет полный доступ ко всему адресному пространству. Системы с общей памятью называют *сильно связанными* (closely coupled systems). Подобное построение вычислительных систем имеет место как в классе SIMD, так и в массе MIMD. Иногда, чтобы подчеркнуть это обстоятельство, вводят специальные подклассы, используя для их обозначения аббревиатуры SM-SIMD (Shared Memory SIMD) и SM-MIMD (Shared Memory MIMD).

В варианте *с распределенной памятью* каждому из процессоров придается собственная память. Процессоры объединяются в сеть и могут при необходимости обмениваться данными, хранящимися в их памяти, передавая друг другу так называемые *сообщения*. Такой вид ВС называют *слабо связанными* (loosely coupled systems). Слабо связанные системы также встречаются как в классе SIMD, так и в классе MIMD, и иной раз, чтобы подчеркнуть данную особенность, вводят подклассы DM-SIMD (Distributed Memory SIMD) и DM-MIMD (Distributed Memory MIMD).

В некоторых случаях вычислительные системы с общей памятью называют *мультипроцессорами,* а системы с распределенной памятью — *мультикомпьютерами*.

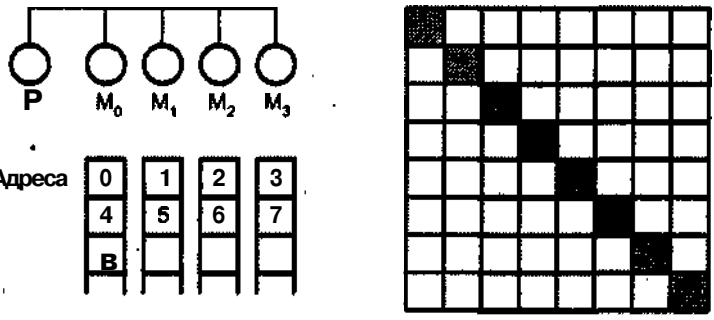
Различие между общей и распределенной памятью - это разница в структуре виртуальной памяти, то есть в том, как память выгладит со стороны процессора Физически почти каждая система памяти разделена на автономные компоненты, доступ *к* которым может производиться независимо. Общую память от распределенной отличает то, каким образом подсистема памяти интерпретирует поступивший от процессора адрес ячейки. Для примера положим, что процессор выполняет команду Load RO , i, означающую "Загрузить регистр R0 содержимым ячейки i ". В случае общей памяти i - это глобальный адрес, и для любого процессора указывает на одну и ту же ячейку. В распределенной системе памяти i - это локальный адрес. Если два процессора выполняют команду load RO, i, то каждый из них обращается к i-й ячейке в своей локальной памяти, то есть к разным ячейкам, и в регистры R0 могут быть загружены неодинаковые значения.

Различие между двумя системами памяти должно учитываться программистом, поскольку оно определяет способ взаимодействия частей распараллеленной программы. В варианте с общей памятью достаточно создать в памяти структуру данных и передавать в параллельно используемые подпрограммы ссылки на эту структуру. В системе с распределенной памятью необходимо в каждой локальной памяти иметь копию совместно используемых данных. Эти копии создаются путем вкладывания разделяемых данных в сообщения, посылаемые другим процессорам.

Память с чередованием адресов

Физически память вычислительной системы состоит из нескольких модулей (банков), при этом существенным вопросом является то, как в этом случае распределено адресное пространство (набор всех адресов, которые может сформировать процессор). Один из способов распределения виртуальных адресов по модулям памяти состоит в разбиении адресного пространства на последовательные блоки. Если память состоит из *n* банков, то ячейка с адресом i при поблочном разбиении будет находиться в банке с номером *i/n*. В системе *памяти с чередованием адресов* (interleaved memory) последовательные адреса располагаются в различных банках: ячейка с адресом i находится в банке с номером *i* mod *n*. Пусть, например, память состоит из четырех банков, по 256 байт в каждом. В схеме, ориентированной на блочную адресацию, первому банку будут выделены виртуальные адреса 0-255, второму - 256-511 и т. д. В схеме с чередованием адресов последовательные ячейки в первом банке будут иметь виртуальные адреса 0, 4, 8, ..,, во втором банке — 1, 5, 9 и т. д. (рис. 11.1, а).

Распределение адресного пространства по модулям дает возможность одновременной обработки запросов на доступ к памяти, если соответствующие адреса относятся к разным банкам. Процессор может в одном из циклов затребовать доступ к ячейке *i,* а в следующем цикле - к ячейке j Если *i u j* находятся в разных банках, информация будет передана в последовательных циклах.. Здесь под циклом понимается цикл процессора, в то время как полный цикл памяти занимает несколько циклов процессора. Таким образом, в данном случае процессор не должен ждать, пока будет завершен полный цикл обращения к ячейке *i.* Рассмотренный прием позволяет повысить пропускную способность: если система памяти состоит из



а *б*

Рис. 11.1. Память с чередованием адресов: а - распределение адресов; *б* - элементы, извлекаемые с шагом 9 из массива 8x8

достаточного числа банков, имеется возможность обмена информацией между процессором и памятью со скоростью одно слово за цикл процессора, независимо от длительности цикла памяти.

Решение о том, какой вариант распределения адресов выбрать (поблочный или с расслоением), зависит от ожидаемого порядка доступа к информации. Программы компилируются так, что последовательные команды располагаются в ячейках с последовательными адресами, поэтому высока вероятность, что после команды, извлеченной из ячейки с адресом *i,* будет выполняться команда из ячейки i + 1. Элементы векторов компилятор также помещает в последовательные ячейки, поэтому в операциях с векторами можно использовать преимущества метода чередования. По этой причине в векторных процессорах обычно применяется какой-либо вариант чередования адресов. В мультипроцессорах с совместно используемой памятью тем не менее используется поблочная адресация, поскольку схемы обращения к памяти в MIMD-системах могут сильно различаться. В таких системах целью является соединить процессор с блоком памяти и задействовать максимум находящейся в нем информации, прежде чем переключиться на другой блок памяти.

Системы памяти зачастую обеспечивают дополнительную гибкость при извлечении элементов векторов. В некоторых системах возможна одновременная загрузка каждого n-го элемента вектора, например, при извлечении элементов вектора *v1* хранящегося в последовательных ячейках памяти при и - 4, память возвратит vO, *v4, v*8. Интервал между элементами называют *шагом по индексу* или *«страйДомъ* (stride). Одним из интересных применений этого свойства может служить доступ к матрицам. Если шаг по индексу на единицу больше числа строк в матрице, одиночный запрос на доступ к памяти возвратит все диагональные элементы матрицы (рис. 11.1, *б).* Ответственность за то, чтобы все извлекаемые элементы матрицы располагались в разных банках, ложится на программиста.

Модели архитектуры памяти вычислительных систем

В рамках как совместно используемой, так и распределенной памяти реализуется несколько моделей архитектур системы памяти.

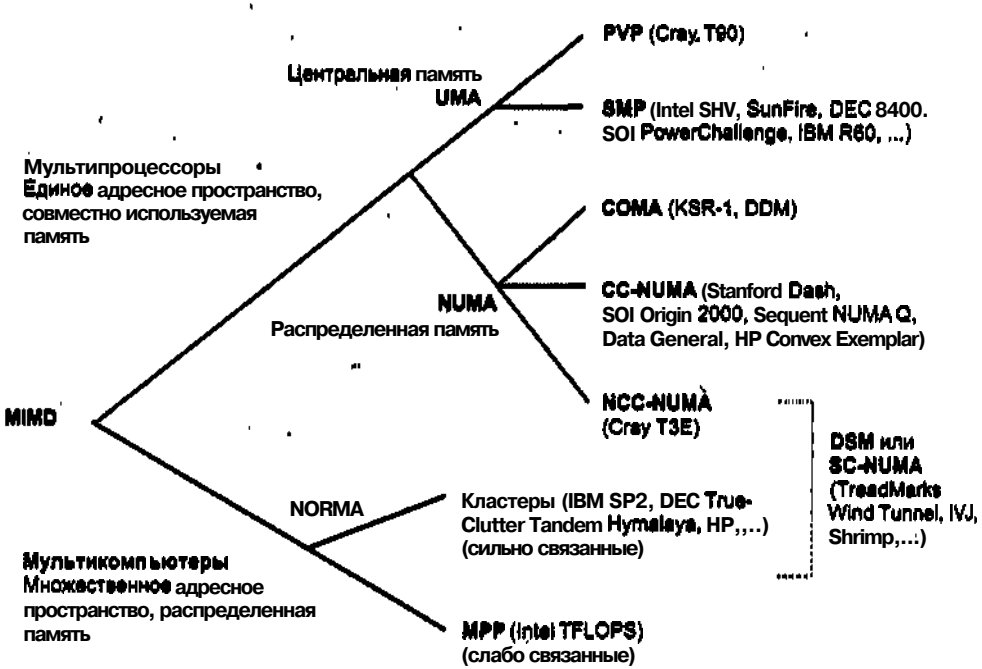


Рис. 11.2. Классификация моделей архитектур памяти вычислительных систем

На рис. 11.2 приведена классификация таких моделей, применяемых в вычислительных системах класса MIMD (верна и для класса SIMD).

Модели архитектур совместно используемой памяти

В системах с общей памятью все процессоры имеют равные возможности по доступу к единому адресному пространству. Единая память может быть построена как одноблочная или по модульному принципу, но обычно практикуется второй вариант.

Вычислительные системы с общей памятью, где доступ любого процессора к памяти производится единообразно и занимает одинаковое время, называют *системами с однородным доступом к памяти и обозначают аббревиатурой* UMA (Uniform Memory Access). Это наиболее распространенная архитектура памяти параллельных ВС с общей памятью [120].

Технически UMA-системы предполагают наличие узла, соединяющего каждый из *n* процессоров с каждым из *m* модулей памяти. Простейший путь построения таких ВС - объединение нескольких процессоров *(P)* с единой памятью (M_p) посредством общей шины - показан на рис. 11.3, *а.* В этом случае, однако, в каждый момент времени обмен по шине может вести только один из процессоров, то есть процессоры должны соперничать за доступ к шине. Когда процессор P_i, выбирает из памяти команду, остальные процессоры P_j *(i не равно j)* должны ожидать, пока шина освободится. Если в систему входят только два процессора, они в состоянии работать с производительностью, близкой к максимальной, поскольку их доступ к шине

можно чередовать; пока один процессор декодирует и выполняет команду, другой вправе использовать шину для выборки из памяти следующей команды. Однако когда добавляется третий процессор, производительность начинает падать. При наличии на шине десяти процессоров кривая быстродействия шины (рис. 11.3, *в)* становится горизонтальной, так что добавление 11-го процессора уже не дает повышения производительности. Нижняя кривая на этом рисунке иллюстрирует тот факт, что память и шина обладают фиксированной пропускной способностью, определяемой комбинацией длительности цикла памяти и протоколом шины, и в многопроцессорной системе с общей шиной эта пропускная способность распределена между несколькими процессорами. Если длительность цикла процессора больше по сравнению с циклом памяти, к шине можно подключать много процессоров. Однако фактически процессор обычно намного быстрее памяти, поэтому данная схема широкого применения не находит.

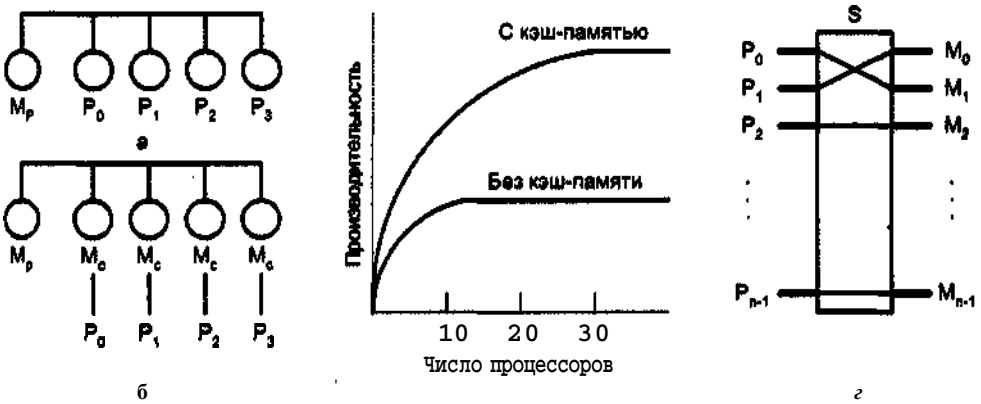


Рис. 11.3. Общая память: *а* — объединение процессоров с помощью шины; б — система с локальными кэшами; *в* - производительность системы как функция от числа процессоров на Шине; г — многопроцессорная ВС с общей памятью, состоящей из отдельных модулей

Альтернативный способ построения многопроцессорной ВС,с общей памятью на основе UMA показан на рис. 11.3, *г.* Здесь шина заменена коммутатором, маршрутизирующим запросы процессора к одному из нескольких модулей памяти. Несмотря на то что имеется несколько модулей памяти, все они входят в единое виртуальное адресное пространство. Преимущество такого подхода в том, что коммутатор в состоянии параллельно обслуживать несколько запросов. Каждый процессор может быть соединен со своим модулем памяти и иметь доступ к нему на максимально допустимой скорости. Соперничество между процессорами может возникнуть при попытке одновременного доступа к одному и тому же модулю памяти, В этом случае доступ получает только один процессор, а прочие — блокируются.

К сожалению, архитектура UMA не очень хорошо масштабируется. Наиболее распространенные системы содержат 4-8 процессоров, значительно реже 32-64 процессора. Кроме того, подобные системы нельзя отнести к отказоустойчивым, так как отказ одного процессора или модуля памяти влечет отказ всей ВС.

Другим подходом к построению ВС с общей памятью является *неоднородный доступ к памяти,* обозначаемый как NUMA (Non-Uniform Memory Access). Здесь по-прежнему фигурирует единое адресное пространство, но каждый процессор имеет локальную память. Доступ процессора к собственной Локальной памяти производится напрямую, что намного быстрее, чем доступ к удаленной памяти через коммутатор или сеть. Такая система может быть дополнена глобальной памятью, тогда локальные запоминающие устройства играют роль быстрой кэш-памяти для глобальной памяти. Подобная схема может улучшить производительность ВС, но не в состоянии неограниченно отсрочить выравнивание прямой производительности. При наличии у каждого процессора локальной кэш-памяти (рис. 11.3,6) существует высокая вероятность ($p > 0{,}9$) того, что нужные команда или данные уже находятся в локальной памяти. Разумная вероятность попадания в локальную память существенно уменьшает число обращений процессора к глобальной памяти и, таким образом, ведет к повышению эффективности. Место излома кривой производительности (верхняя кривая на рис. 11.3, *в),* соответствующее точке, в которой добавление процессоров еще остается эффективным, теперь перемещается в область 20 процессоров, а точка, где кривая становится горизонтальной, - в область 30 процессоров.

В рамках концепции *NUMA* реализуется несколько различных подходов, обозначаемых аббревиатурами *COMA, CC-NUMA* и *NCC-NUMA.*

В *архитектуре только с кэш-памятью* (COMA, Cache Only Memory Architecture) локальная память каждого процессора построена как большая кэш-память для быстрого доступа со стороны «своего» процессора [44]. Кэши всех процессоров в совокупности рассматриваются как глобальная память системы. Собственно глобальная память отсутствует. Принципиальная особенность концепции COMA выражается в динамике. Здесь данные не привязаны статически к определенному модулю памяти и не имеют уникального адреса, остающегося неизменным в течение всего времени существования переменной. В архитектуре COMA данные переносятся в кэш-память того процессора, который последним их запросил, при этом переменная не фиксирована уникальным адресом и в каждый момент времени может размещаться в любой физической ячейке. Перенос данных из одного локального кэша в другой не требует участия в этом процессе операционной системы, но подразумевает сложную и дорогостоящую аппаратуру управления памятью. Для организации такого режима используют так называемые *каталоги кэшей.* Отметим также, что последняя копия элемента данных никогда из кэш-памяти не удаляется.

Поскольку в архитектуре COMA данные перемещаются в локальную кэш-память процессора-владельца, такие ВС в плане производительности обладают существенным преимуществом над другими архитектурами NUMA. С другой стороны, если единственная переменная или две различные переменные, хранящиеся в одной строке одного и того же кэша, требуются двум процессорам, эта строка кэша должна перемещаться между процессорами туда и обратно при каждом доступе к данным. Такие эффекты могут зависеть от деталей распределения памяти и приводить к непредсказуемым ситуациям.

*Модель кэш-когерентного доступа к неоднородной памяти (*CC-NUMA, Cache Coherent Non-Uniform Memory Architecture) принципиально отличается от модели

COMA. В системе CC-NUMA используется не кэш-память, а обычная физически распределенная память. Не происходит никакого копирования страниц или данных между ячейками памяти. Нет никакой программно реализованной передачи сообщений. Существует просто одна карта памяти, с частями, физически связанными медным кабелем, и «умные» аппаратные средства. Аппаратно реализованная кэш-когерентность означает, что не требуется какого-либо программного обеспечения для сохранения множества копий обновленных данных или их передачи. Со всем этим справляется аппаратный уровень. Доступ к локальным Модулям памяти в разных узлах системы может производиться одновременно и происходит быстрее, чем к удаленным модулям памяти.

Отличие модели с кэш-некогерентным доступом к неоднородной памяти (NCC-NUMA, Non-Cache Coherent Non-Uniform Memory Architecture) от CC-NUMA очевидно из названия. Архитектура памяти предполагает единое адресное пространство, но не обеспечивает согласованности глобальных данных на аппаратном уровне. Управление использованием таких данных полностью возлагается на программное обеспечение (приложения или компиляторы). Несмотря на это обстоятельство, представляющееся недостатком архитектуры, она оказывается весьма полезной при повышении производительности вычислительных систем с архитектурой памяти типа DSM, рассматриваемой в разделе «Модели архитектур распределенной памяти».

В целом, ВС с общей памятью, построенные по схеме NUMA, называют *архитектурами с виртуальной общей памятью* (virtual shared memory architectures). Данный вид архитектуры, в частности CC-NUMA, в последнее время рассматривается как самостоятельный и довольно перспективный вид вычислительных систем класса MIMD, поэтому такие ВС ниже будут обсуждены более подробно. •

Модели архитектур распределенной памяти

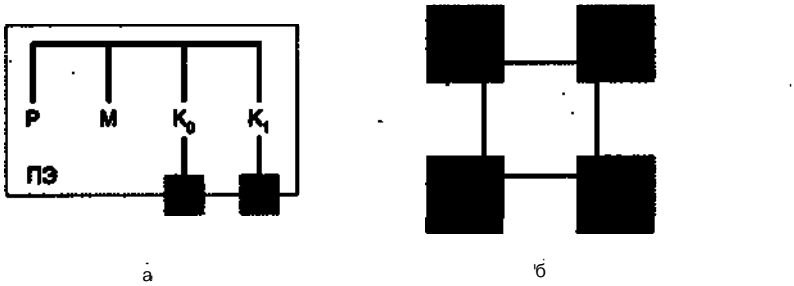
В системе с распределенной памятью каждый процессор обладает собственной памятью и способен адресоваться только к ней. Некоторые авторы называют этот тип *систем многомашинными ВС или мультикомпьютерами,* подчеркивая тот факт, что блоки, из которых строится система, сами по себе являются небольшими вычислительными системами с процессором и памятью. Модели архитектур с распределенной памятью принято обозначать как *архитектуры без прямого доступа к удаленной памяти* (NORMA, No Remote Memory Access). Такое название следует из того факта, что каждый процессор имеет доступ только к своей локальной памяти. Доступ к удаленной памяти (локальной памяти другого процессора) возможен только путем обмена сообщениями с процессором, которому принадлежит адресуемая память.

Подобная организация характеризуется рядом достоинств. Во-первых, при доступе к данным не возникает конкуренции за шину или коммутаторы — каждый процессор может полностью использовать полосу пропускания тракта связи с собственной локальной памятью. Во-вторых, отсутствие общей шины означает, что нет и связанных с этим ограничений на число процессоров: размер системы ограничивает только сеть, объединяющая процессоры. В-третьих, снимается проблема когерентности кэш-памяти. Каждый процессор вправе самостоятельно менять свои данные, не заботясь о согласовании копий данных в собственной локальной кэш-памяти с кэшами других процессоров.

Основной недостаток ВС с распределенной памятью заключается в сложности обмена информацией между процессорами. Если какой-то из процессоров нуждается в данных из памяти другого процессора, он должен обменяться с этим процессором сообщениями. Это приводит к двум видам издержек:

- требуется время для того, чтобы сформировать и переслать сообщение от одного процессора к другому;
- для обеспечения реакции на сообщения от других процессоров принимающий процессор должен получить запрос прерывания и выполнить процедуру обработки этого прерывания.

Структура системы с распределенной памятью приведена на рис. 11.4. В левой части (рис. 11.4, а) показан один процессорный элемент (ПЭ). Он включает в себя собственно процессор (Р), локальную память (М) и два контроллера ввода/вывода (K_o и K i), В правой части (рис. 11.4, *б)* показана четырехпроцессорная система, иллюстрирующая, каким образом сообщения пересылаются от одного процессора к другому. По отношению к каждому ПЭ все остальные процессорные элементы можно рассматривать просто как устройства ввода/вывода. Для посылки сообщения в другой ПЭ процессор формирует блок данных в своей локальной памяти и извещает свой локальный контроллер о необходимости передачи информации на внешнее устройство. По сети межсоединений это сообщение пересылается на приемный контроллер ввода/вывода принимающего ПЭ. Последний находит место для сообщения в собственной локальной памяти и уведомляет процессор-источник о получении сообщения.



Рис, 11 .4. Вычислительная система с распределенной памятью: *а* - процессорный элемент; *б* - объединение процессорных элементов

Интересный вариант системы с распределенной памятью представляет собой модель *распределенной совместно используемой памяти (DSM, Distribute Shared* Memory), известной также и под другим названием *архитектуры с неоднородным доступом к памяти и программным обеспечением когерентности (SC*-NUMA, Software-Coherent Non-Uniform Memory Architecture). Идея этой модели состоит в том, что ВС, физически будучи системой с распределенной памятью, благодаря операционной системе представляется пользователю как система с общей памятью. Это означает, что операционная система предлагает пользователю единое адресное пространство, несмотря на то что фактическое обращение к памяти "чужого" компьютера ВС по-прежнему обеспечивается путем обмена сообщениями.

Мультипроцессорная когерентность кэш-памяти

Мультипроцессорная система с разделяемой памятью состоит из двух или более независимых процессоров, каждый из которых выполняет либо часть большой программы, либо независимую программу. Все процессоры обращаются к командам и данным, хранящимся в общей основной памяти. Поскольку память является обобществленным ресурсом, при обращении к ней между процессорами возникает соперничество, в результате чего средняя задержка на доступ к памяти увеличивается. Для сокращения такой задержки каждому процессору придается локальная кэш-память, которая, обслуживая локальные обращения к памяти, во многих случаях предотвращает необходимость доступа к совместно используемой основной памяти. В свою очередь, оснащение каждого процессора локальной кэш-памятью приводит к так называемой проблеме когерентности или обеспечения согласованности кэш-памяти. Согласно [93, 215], система является когерентной, если каждая операция чтения по какому-либо адресу, выполненная любым из процессоров, возвращает значение, занесенное в ходе последней операции записи по этому адресу, вне зависимости от того, какой из процессоров производил запись последним.

В простейшей форме проблему когерентности кэш-памяти можно пояснить следующим образом (рис 11.5). Пусть два процессора P1 и P_2 связаны с общей памятью посредством шины. Сначала обаС процессора читают переменную *x*. Копии блоков, содержащих эту переменную, пересылаются из основной памяти в локальные кэши обоих процессоров (рис. 11.5, а). Далее процессор P_1 выполняет операцию увеличения значения переменной x на единицу. Так как копия переменной уже находится в кэш-памяти данного процессора, произойдет кэш-попадание и значение *x* будет изменено только в кэш-памяти 1. Если теперь процессор P_2 вновь выполнит операцию чтения *x,* то также произойдет кэш-попадание и P_2 получит хранящееся в его кэш-памяти «старое» значение *x* (рис. 11.5,5).

Поддержание согласованности требует, чтобы при изменении элемента данных одним из процессоров соответствующие изменения были проведены в кэш-памяти остальных процессоров, где есть копия измененного элемента данных, а также в общей памяти. Схожая проблема возникает, кстати, и в однопроцессорных системах, где присутствует несколько уровней кэш-памяти. Здесь требуется согласовать содержимое кэшей разных уровней.

В решении проблемы когерентности выделяются два подхода: программный и аппаратный. В некоторых системах применяют стратегии, совмещающие оба подхода.

Программные способы решения проблемы когерентности

Программные приемы решения проблемы когерентности позволяют обойтись без дополнительного оборудования или свести его к минимуму [72]. Задача возлагается на компилятор и операционную систему. Привлекательность такого подхода в возможности устранения некогерентности еще до этапа выполнения программы,

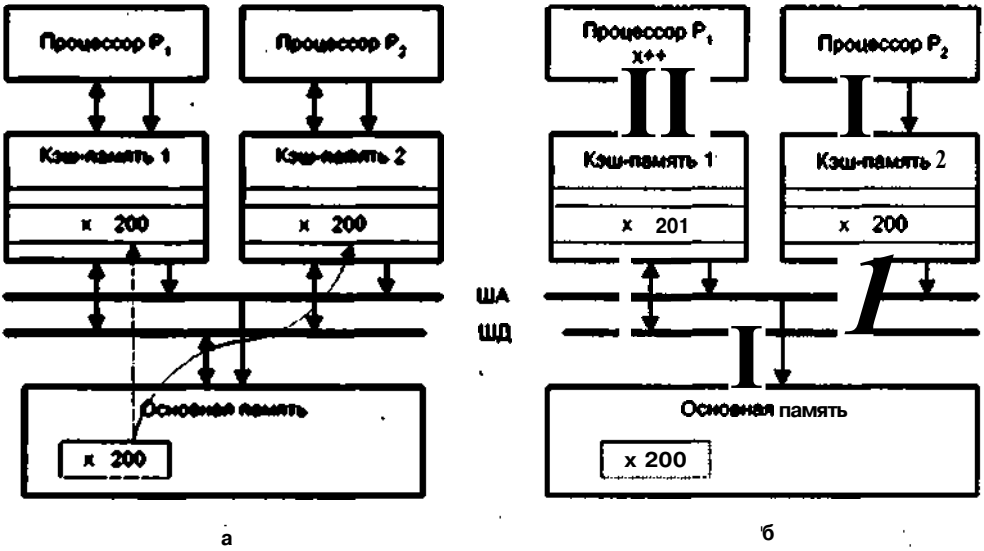


Рис. 11.5. Иллюстрация проблемы когерентности памяти: *а* - содержимое памяти до изменения значения x; *б* — после изменения

однако принятые компилятором решения могут в целом отрицательно сказаться на эффективности кэш-памяти.

Компилятор анализирует программный код, определяет те совместно используемые данные, которые могут стать причиной некогерентности, и помечает их. В процессе выполнения программы операционная система или соответствующая аппаратура предотвращают кэширование (занесение в кэш-память) помеченных данных, и в дальнейшем для доступа к ним, как при чтении, так и при записи, приходится обращаться к «медленной» основной памяти. Учитывая, что некогерентность возникает только в результате операций записи, происходящих значительно реже, чем чтение, рассмотренный прием следует признать недостаточно удачным.

Более эффективными представляются способы, где в ходе анализа программы определяются безопасные периоды использования общих переменных и так называемые критические периоды, где может проявиться некогерентность. Затем компилятор вставляет в генерируемый код инструкции, позволяющие обеспечить когерентность кэш-памятей именно в такие критические периоды.

Аппаратные способы решения проблемы когерентности

Большинство из предложенных способов борьбы с некогерентностью ориентированы на динамическое (в процессе вычислений) распознавание и устранение несогласованности копий совместно используемых данных с помощью специальной аппаратуры. Аппаратные методы обеспечивают более высокую производительность, поскольку издержки, связанные с некогерентностью, имеют место только при возникновении ситуации некогерентности. Кроме того, непрограммный подход прозрачен для программиста и пользователя [215]. Аппаратные механизмы

преодоления проблемы когерентности принято *называть протоколами когерентности кэш-памяти*.

Как известно, для обеспечения идентичности копий данных в кэше и основной памяти в однопроцессорных системах применяется одна из двух стратегий: *сквозная запись* (write through) или *обратная запись* (write back). При сквозной записи новая информация одновременно заносится как в кэш, так и в основную память. При обратной записи все изменения производятся только в кэш-памяти, а обновление содержимого основной памяти происходит лишь при удалении блока из кэш-памяти путем пересылки удаляемого блока в соответствующее место основной памяти. В случае мультипроцессорной системы, когда копии совместно используемых данных могут находиться сразу в нескольких кэшах, необходимо обеспечить когерентность всех копий. Ни сквозная, ни обратная запись не предусматривают такой ситуации, и для ее разрешения опираются на другие приемы, а именно: *запись с аннулированием (*write inv*alidate)* и *запись с обновлением (*write *update)*. *По*следняя известна также под названием *записи с трансляцией* (write broadcast).

В варианте записи с аннулированием, если какой-либо процессор производит изменения в одном из блоков своей кэш-памяти, все имеющиеся копии этого блока в других локальных кэшах аннулируются, то есть помечаются как недостоверные. Для этого бит достоверности измененного блока (строки) во всех прочих кэшах устанавливается в 0. Идею записи с аннулированием иллюстрирует рис. 11.6, где показано исходное состояние системы памяти, где копия переменной *x* имеется во всех кэшах (рис. *11.6, а), а* также ее состояние после записи нового значения *x* в кэш с номером 2 (рис 11.6, *б)*.

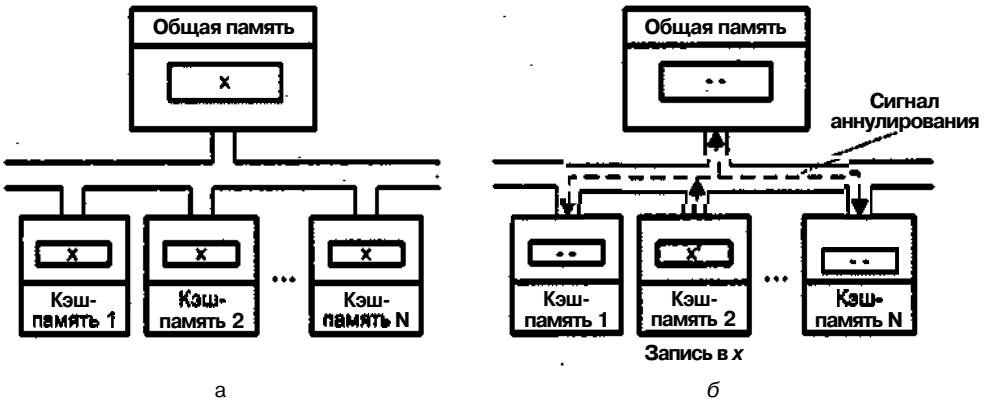


Рис. 11.6. Запись с аннулированием: а - исходное состояние; *6* - после изменения значений x в кэш-памяти 2

Если впоследствии другой процессор попытается прочитать данные из своей копии такого блока, произойдет кэш-промах. Следствием кэш-промаха должно быть занесение в локальную кэш-память читающего процессора корректной копии блока. Некоторые схемы когерентности позволяют получить корректную копию непосредственно из той локальной кэш-памяти, где блок подвергся модификации. Если такая возможность отсутствует, новая копия берется из основной памяти. В слу-

чае сквозной записи это может быть сделано сразу же, а при использовании обратной записи модифицированный блок предварительно должен быть переписан в основную память.

Запись с обновлением предполагает, что любая запись в локальный кэш немедленно дублируется и во всех остальных кэшах, содержащих копию измененного блока (немедленное обновление блока в основной памяти не является обязательным). Этот случай иллюстрирует рис. 11.7.

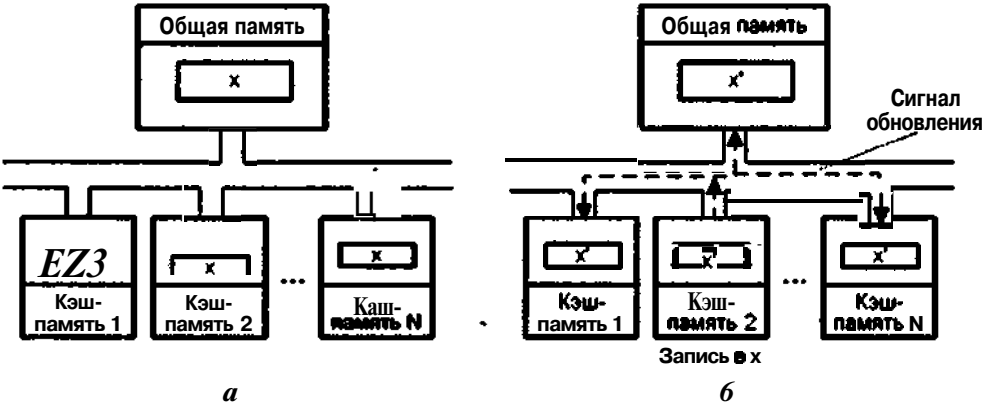


Рис. 11.7. Запись с обновлением: *в* - исходное состояние; *б* - после изменения значения x в кэш-памяти 2

Стратегия записи с обновлением требует широковещательной передачи новых данных по сети межсоединений, что осуществимо не при любой топологии сети.

В общем случае для поддержания когерентности в мультипроцессорных системах имеются следующие возможности:

- совместно используемая кэш-память;
- некэшируемые данные;
- широковещательная запись;
- протоколы наблюдения;
- протоколы на основе справочника.

Совместно используемая кэш-память. Первое и наиболее простое решение - вообще отказаться от локальных кэшей и все обращения к памяти адресовать к одной общей кэш-памяти, связанной со всеми процессорами посредством какой-либо коммуникационной сети. Хотя данный прием обеспечивает когерентность копий данных и прозрачен для пользователя, количество конфликтов по доступу к памяти он не снижает, поскольку возможно одновременное обращение нескольких процессоров к одним и тем же данным в обшей кэш-памяти. Кроме того, наличие разделяемой кэш-памяти нарушает важнейшее условие высокой производительности, согласно которому процессор и кэш-память должны располагаться как можно ближе друг к другу. Положение осложняется и тем, что каждый доступ к кэшу связан с обращением к арбитру, который определяет, какой из процессоров получит дос-

туп к кэш-памяти. Тем не менее общая задержка обращения к памяти в целом уменьшается.

Некэшируемые данные. Проблема когерентности имеет отношение к тем данным, которые в ходе выполнения программы могут быть изменены. Одно из вероятных решений — это запрет кэширования таких данных. Технически запрет на кэширование отдельных байтов и слов достаточно трудно реализуем. Несколько проще сделать некэшируемым определенный блок данных. При обращении процессора к такому блоку складывается ситуация кэш-промаха, производится доступ к основной памяти, но копия блока в кэш не заносится. Для реализации подобного приема каждому блоку в основной памяти должен быть придан признак, указывающий, является ли блок кэшируемым или нет.

Если кэш-система состоит из раздельных кэшей команд и данных, сказанное относится главным образом к кэш-памяти данных, поскольку современные подходы к программированию не рекомендуют модификацию команд программы. Следовательно, по отношению к информации в кэше команд применяется только операция чтения, что не влечет проблемы когерентности.

В отношении того, какие данные не должны кэшироваться, имеется несколько подходов.

В первом варианте запрещается занесение в кэш лишь той части совместно используемых данных, которая служит для управления критическими секциями программы, то есть теми частями программы, где процессоры могут изменять разделяемые ими данные. Принятие решения о том, какие данные могут кэшироваться, а какие — нет, возлагается на программиста, что делает этот способ непрозрачным для пользователя.

Во втором случае накладывается запрет на кэширование всех совместно используемых данных, которые в процессе выполнения программы могут быть изменены. Естественно, что для доступа к таким данным приходится обращаться к медленной основной памяти и производительность процессора падает. На первый взгляд, в варианте, где запрещается кэширование только управляющей информации, производительность процессора будет выше, однако, прежде чем сделать такой вывод, нужно учесть одно обстоятельство. Дело в том, что для сохранения согласованности данных, модифицируемых процессором в ходе выполнения критической секции программы, строки с копиями этих данных в кэш-памяти при выходе из критической секции нужно аннулировать. Данная операция носит название *очистки кэш-памяти* (cache flush). Очистка необходима для того, чтобы к моменту очередного входа в критическую секцию в кэш-памяти не осталось "устаревших"- данных. Регулярная очистка кэша при каждом выходе из критической секции снижает производительность процессора за счет увеличения времени, нужного для восстановления копий в кэш-памяти. Ситуацию, можно несколько улучшить, если вместо очистки всей кэш-памяти помечать те блоки, к которым при выполнении критической секции было обращение, тогда при покидании критической секции достаточно очищать только эти помеченные блоки.

Широковещательная запись. При широковещательной записи каждый запрос на запись в конкретную кэш-память направляется также и всем остальным кэшам системы. Это заставляет контроллеры кэшей проверить, нет ли там копии изменяе-

мого блока. Если такая копия найдена, то она аннулируется или обновляется, в зависимости от применяемой схемы. Метод широковещательной записи связан с дополнительными групповыми операциями с памятью (транзакциями), поэтому он реализован лишь в больших вычислительных системах.

На двух последних возможностях поддержания когерентности в мультипроцессорных системах остановимся более подробно.

Протоколы наблюдения

В *протоколах наблюдения* (snoopy protocols или просто snooping) ответственность за поддержание когерентности всех кэшей многопроцессорной системы возлагается на контроллеры кэшей. В системах, где реализованы протоколы наблюдения, контроллер каждой локальной кэш-памяти содержит *блок слежения за шиной* (рис. 11.8), который следит за всеми транзакциями на общей шине и, в частности, контролирует все операции записи. Процессоры должны широковещательно передавать на шину любые запросы на доступ к памяти, потенциально способные изменить состояние когерентности совместно используемых блоков данных. Локальный контроллер кэш-памяти каждого процессора затем определяет, присутствует ли в его кэш-памяти копия модифицируемого блока, и если это так, то такой блок аннулируется или обновляется.

Протоколы наблюдения характерны для мультипроцессорных систем на базе шины, поскольку общая шина достаточно просто обеспечивает как наблюдение, так и широковещательную передачу сообщений. Однако здесь необходимо принимать меры, чтобы повышенная нагрузка на шину, связанная с наблюдением и трансляцией сообщений, не «съела» преимуществ локальных кэшей.

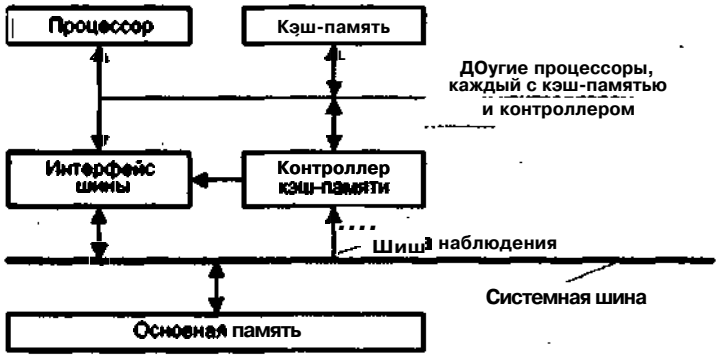


Рис. 11.8. Кэш-память с контроллером наблюдения за шиной

Ниже рассматриваются некоторые из наиболее распространенных протоколов наблюдения. Большинство из них описываются упрощенно, а их детальное изложение можно найти по ссылкам на литературные источники.

В большинстве протоколов стратегия обеспечения когерентности кэш-памяти расценивается как смена состояний в конечном автомате. При таком подходе предполагается, что любой блок в локальной кэш-памяти может находиться в одном из фиксированных состояний. Обычно число таких состояний не превышает четырех, поэтому для каждой строки кэш-памяти в ее теге имеются два бита, называе-

мые битами состояния (SB, Status Bit). Следует также учитывать, что некоторым идентичным по смыслу состояниям строки кэша разработчиками различных протоколов присвоены разные наименования. Например, состояние строки, в которой были произведены локальные изменения, в одних протоколах называют *Dirty* («грязный»), а в других - Modified («модифицированный» или «измененный»).

Протокол сквозной записи. Этот протокол представляет собой расширение стандартной процедуры сквозной записи, известной по однопроцессорным системам. В нем запись в локальную кэш-память любого процессора сопровождается записью в основную память. В дополнение, все остальные кэши, содержащие копию измененного блока, должны объявить свою копию недействительной. Протокол считается наиболее простым, но при большом числе процессоров приводит к значительному трафику шины, поскольку требует повторной перезагрузки измененного блока в те кэши, где этот блок ранее был объявлен недействительным [211]. Кроме того, производительность процессоров при записи в совместно используемые переменные может упасть из-за того, что для продолжения вычислений процессоры должны ожидать, пока завершатся все операции записи [141].

Протокол обратной записи. В основе протокола лежит стандартная схема обратной записи, за исключением того, что расширено условие перезаписи блока в основную память. Так, если копия блока данных в одном из локальных кэшей подверглась модификации, этот блок будет переписан в основную память при выполнении одного из двух условий:

- блок удаляется из той кэш-памяти, где он был изменен;
- другой процессор обратился к своей копии измененного блока.

Если содержимое строки в локальном кэше не модифицировалось, перезапись в основную память не производится. Доказано, что такой протокол по эффективности превосходит схему сквозной записи, поскольку необходимо переписывать только измененные блоки [211].

Несмотря на более высокую производительность, протокол обратной записи также не идеален, так как решает проблему когерентности лишь частично. Когда процессор обновляет информацию в своей кэш-памяти, внесенные изменения не наблюдаемы со стороны других процессоров до момента перезаписи измененного блока в основную память, то есть другие процессоры не знают, что содержимое по данному адресу было изменено, до тех пор пока соответствующая строка не будет переписана в основную память. Эта проблема часто решается путем наложения условия, что кэши, которые собираются изменить содержимое совместно используемого блока, должны получить эксклюзивные права на этот блок, как это делается в рассматриваемом позже протоколе Berkeley [141].

В работе [110] приводятся результаты сравнения среднего трафика шины для протоколов обратной и сквозной записи. Обнаружено, что когда коэффициент кэш-попаданий приближается к 100%, протокол обратной записи вообще не требует трафика шины, так как все необходимые строки находятся в кэш-памяти. В свою очередь, протоколу сквозной записи необходим, по крайней мере, один цикл шины на каждую операцию чтения, поскольку предыдущая операция записи могла аннулировать копию данных в локальном кэше. В работе также доказано, что применение протокола обратной записи взамен протокола сквозной записи способно

снизить трафик шины на 50%, однако обратная запись по сравнению со сквозной влечет более серьезные проблемы когерентности. Это связано с тем, что даже основная память не всегда содержит последнее значение элемента данных.

Протокол однократной записи. Протокол однократной записи (write-once), предложенный Гудменом.[110], - первый из упоминающихся в публикациях протоколов обеспечения когерентности кэш-памяти. Он относится к схемам на основе наблюдения, действующим на принципе записи с аннулированием. Протокол предполагает, что первая запись в любую строку кэш-памяти производится по схеме сквозной записи, при этом контроллеры других кэшей объявляют свои копии измененного блока недействительными. С этого момента только процессор, произведший запись, обладает достоверной копией данных [141]. Последующие операции записи в рассматриваемую строку выполняются в соответствии с протоколом обратной записи [51].

Основной недостаток протокола в том, что он требует первоначальной записи в основную память, даже если эта строка не используется другими процессорами.

Диаграмма состояний протокола показана на рис. 11.9.

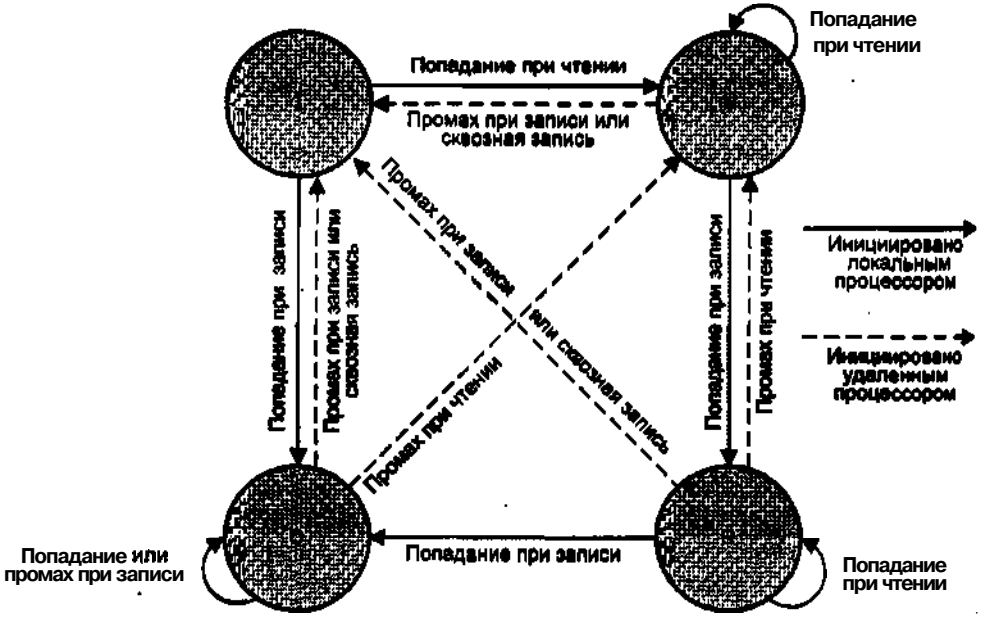


Рис. 11.9. Протокол однократной записи

Для реализации протокола однократной записи каждой строке кэш-памяти приданы два бита. Это позволяет представить четыре состояния, в которых может находиться строка: «недействительная» (I, Invalid), «достоверная» (V, Valid), «резервированная» (R, Reserved) и «измененная» (D, Dirty). В состоянии I строка кэш-памяти не содержит достоверных данных. В состоянии V строка кэша содержит данные, считанные из основной памяти и к данному моменту еще не измененные, то есть строка кэша и блок основной памяти согласованы. Состояние R означает, что с момента считывания из основной памяти в блоке локальной кэш-памяти было

произведено только одно изменение, причем оно учтено и в основной памяти. В состоянии R содержимое строки кэша и основной памяти также является согласованным. Наконец, статус D показывает, что строка кэш-памяти модифицировалась более чем один раз и последние изменения еще не переписаны в основную намять. В этом случае строка кэша и содержимое основной памяти не согласованы.

В процессе выполнения программ блоки слежения за шиной каждой кэш-памяти проверяют, не совпадает ли адрес ячейки, изменяемой в какой-либо локальной кэш-памяти, с одним из адресов в собственном кэше. Если такое совпадение произошло при выполнении операции записи, контроллер кэша изменяет статус соответствующей строки в своей кэш-памяти на I. Если совпадение обнаружено при выполнении операции чтения, состояние строки не изменяется, за исключением случая, когда строка, проверяемая на совпадение, помечена как R или D. Если строка имеет состояние R, оно изменяется на V. Когда строка кэша отмечена как измененная (D), локальная система запрещает считывание элемента данных из основной памяти и данные берутся непосредственно из локальной кэш-памяти, как из источника наиболее «свежей» информации. Во время того же доступа к шипе или непосредственно после него обновленное значение должно быть переписано в основную память, а состояние строки скорректировано на V.

В протоколе однократной записи когерентность сохраняется благодаря тому, что когда выполняется запись, копии изменяемой строки во всех остальных локальных кэшах объявляются недействительными. Таким образом, кэш, выполняющий операцию записи, становится обладателем единственной достоверной копии (при первой записи в строку такая же копия будет и в основной памяти) [110]. При первой записи строка переводится в состояние R, и если впоследствии такая строка удаляется из кэш-памяти, ее перезапись в основную память не требуется. При последующих изменениях строки она помечается как D и работает протокол обратной записи.

В ранее упоминавшейся работе [110] приводятся результаты сравнения протоколов сквозной и обратной записи также и с протоколом однократной записи. Согласно Гудмену, мультипроцессорная система, состоящая из трех компьютеров PDP-11, каждый из которых имеет множественно-ассоциативную четырехканальную кэш-память емкостью 2048 байт при длине строки в 32 байта, показывает следующие показатели трафика шины: 30,76%, 17,55% и 17,38% для протоколов сквозной, обратной и однократной записи соответственно. Таким образом, показатели протокола однократной записи по сравнению с протоколами сквозной и обратной записи несколько лучше.

Протокол Synapse. Данный протокол, реализованный в отказоустойчивой мультипроцессорной системе Synapse N + 1, представляет собой версию протокола однократной записи, где вместо статуса R используется статус D. Кроме того, переход из состояния 0 в состояние V при промахе, возникшем в ходе чтения данных другим процессором, заменен достаточно громоздкой последовательностью. Связано это с тем, что при первом кэш-промахе чтения запросивший процессор не может получить достоверную копию непосредственно из той локальной кэш-памяти, где произошло изменение данных, и вынужден обратиться напрямую к основной памяти[51,138].

Протокол Berkeley. Протокол Berkeley [141] был применен в мультипроцессорной системе Berkeley, построенной на базе RISC-процессоров.

Снижение издержек, возникающих в результате кэш-промахов, обеспечивается благодаря реализованной в этом протоколе идее прав владения на строку кэша Обычно владельцем прав на все блоки данных считается основная память. Прежде чем модифицировать содержимое строки в своей кэш-памяти, процессор должен получить права владения на данную строку. Эти права приобретаются с помощью специальных операций чтения и записи. Если при доступе к блоку, собственником которого в данный момент не является основная память, происходит кэш-промах, процессор, являющийся владельцем строки, предотвращает чтение из основной памяти и сам снабжает запросивший процессор данными из своей локальной кэш-памяти.

Другое улучшение — введение состояния совместного использования (shared). Когда процессор производит запись в одну из строк своей локальной кэш-памяти, он обычно формирует сигнал аннулирования копий изменяемого блока в других кэшах. В протоколе Berkeley сигнал аннулирования формируется только при условии, что в прочих кэшах имеются такие копии. Это позволяет существенно снизить непроизводительный трафик шины. Возможны следующие сценарии.

Прежде всего, каждый раз, когда какой-либо процессор производит запись в свою кэш-память, изменяемая строка переводится в состояние «измененная, частная» (PD, Private Dirty). Далее, если строка является совместно используемой, на шину посылается сигнал аннулирования, и во всех локальных кэшах, где есть копия данного блока данных, эти копии переводятся в состояние «недействительная» (I, Invalid). Если при записи имел место промах, процессор получает копию блока из кэша текущего хозяина запрошенного блока. Лишь после этих действий процессор производит запись в свой кэш.

При кэш-промахе чтения процессор посылает запрос владельцу блока, с тем чтобы получить наиболее свежую версию последнего, и переводит свою новую копию в состояние «только для чтения» *(RO,* Read Only). Если владельцем строки был другой процессор, он помечает свою копию блока как «разделяемую измененную» (SD, Shared Dirty).

Диаграмма состояний протокола Berkeley показана на рис. 11.10.

Сравнивая протоколы однократной записи и Berkeley, можно отметить следующее. Оба протокола используют стратегию обратной записи, при которой измененные блоки удерживаются в кэш-памяти как можно дольше. Основная память обновляется только при удалении строки из кэша. Верхняя граница общего количества транзакций записи на шине определяется той частью протокола однократной записи, где реализуется сквозная запись, так как последняя стратегия порождает на шине операцию записи при каждом изменении, инициированном процессором [141]. Поскольку первая операция записи в протоколе однократной записи является сквозной, она производится даже если данные не являются совместно используемыми. Это влечет дополнительный трафик шины, который возрастает с увеличением емкости кэш-памяти. Доказано, что протокол однократной записи приводит к большему трафику шины по сравнению с протоколом Berkeley [141].

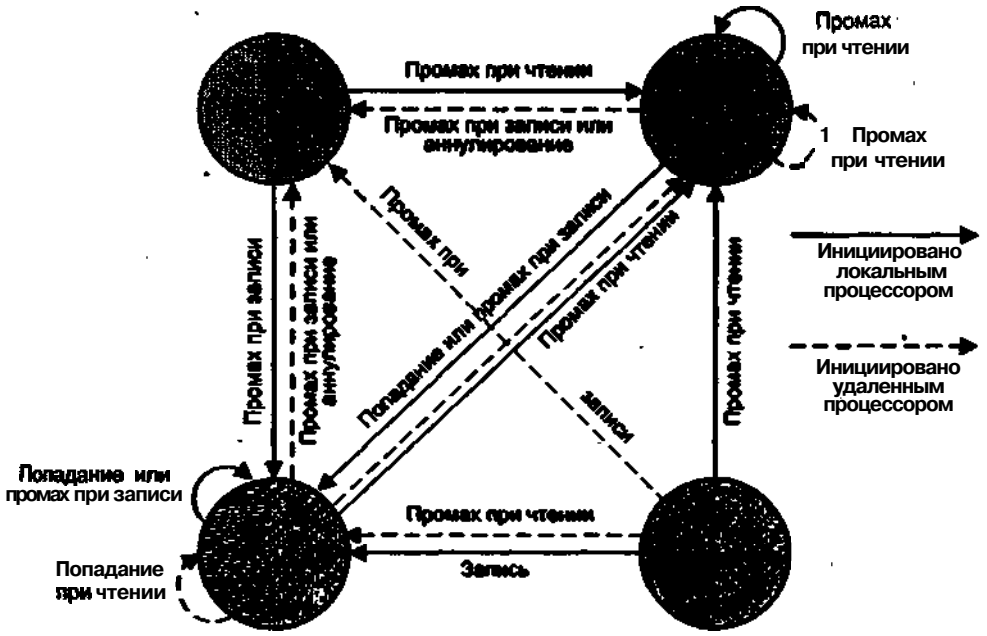


Рис. 11.10. Протокол Berkeley

Для постоянно читаемой и обновляемой строки в протоколе однократной записи необходимы считывание этой строки в кэш, ее локальная модификация в кэше и обратная запись в память. Вся процедура требует двух операций на шине: чтения из основной памяти (ОП) и обратной записи в ОП. С другой стороны, протокол Berkeley исходит из получения прав на строку. Далее блок модифицируется в кэше. Если до удаления из кэша к строке не производилось обращение, число циклов шины будет таким же, как и в протоколе однократной записи. Однако более вероятно, что строка будет запрошена опять, тогда с позиций одиночной кэш-памяти обновление строки кэша нуждается только в одной операции чтения на шине. Таким образом, протокол Berkeley пересылает строки непосредственно между кэшами, в то время как протокол однократной записи передает блок из исходного кэша в основную память, а затем из ОП в запросившие кэши, что имеет следствием общую задержку системы памяти [141],

Протокол Illinois. Протокол Illinois, предложенный Марком Папамаркосом [175], также направлен на снижение трафика шины и, соответственно, времени ожидания процессором доступа к шине. Здесь, как и в протоколе Berkeley, главенствует идея прав владения блоком, но несколько измененная. В протоколе Illinois правом владения обладает любой кэш, где есть достоверная копия блока данных. В этом случае у одного и того же блока может быть несколько владельцев. Когда такое происходит, каждому процессору назначается определенный приоритет и источником информации становится владелец с более высоким приоритетом.

Как и в предыдущем случае, сигнал аннулирования формируется, лишь когда копии данного блока имеются и в других кэшах. Возможные сценарии для протокола Illinois представлены на рис. 11.11,

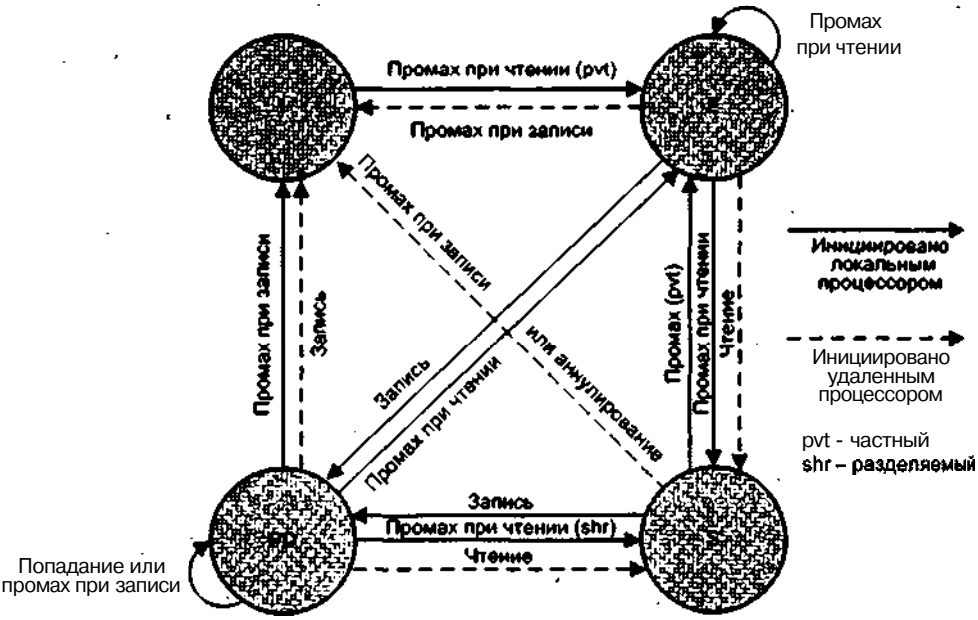


Рис. 11.11. Протокол Illinois

Каждый раз когда какой-либо процессор производит запись в свою кэш-память, изменяемая строка переводится в состояние «измененная частная» (PD, Private Dirty). Если блок данных является совместно используемым, на шину посылается сигнал аннулирования и во всех локальных кэшах, где есть копия данного блока, эти копии переводятся в состояние «недействительная» (I, Invalid). Если при записи случился промах, процессор получает копию из кэша текущего владельца запрошенного блока. Лишь после означенных действий процессор производит запись в свой кэш. Как видно, в этой части имеет место полное совпадение с протоколом Berkeley,

При кэш-промахе чтения процессор посылает запрос владельцу блока, с тем чтобы получить наиболее свежую версию последнего, и переводит свою новую копию в состояние «эксклюзивная»- (E, Exclusive) при условии, что он является единственным владельцем строки. Б противном случае статус меняется на «разделяемая» (S, Shared).

Существенно, что протокол расширяем и тесно привязан как к коэффициенту кэш-промахов, так и к объему данных, которые являются общим достоянием мультипроцессорной системы.

Протокол Firefly. Протокол был предложен Такером и др. **[211]** и реализован в мультипроцессорной системе Firefly Multiprocessor Workstation, разработанной в исследовательском центре Digital Equipment Corporation.

В протоколе Firefly используется запись с обновлением. Возможные состояния строки кэша совпадают с состояниями протокола Illinois (рис. 11,12). Отличие состоит в том, что стратегия обратной записи применяется только к тем строкам, которые находятся в состоянии PD или E, в то время как применительно к строкам

в состоянии S действует сквозная запись. Наблюдающие кэши при обновлении своих копий используют процедуру сквозной записи. Кроме того, наблюдающие кэши, обнаружившие у себя копию строки, возбуждают специальную «разделяемую» линию шины с тем, чтобы записывающий контроллер мог принять решение о том, в какое состояние переводить строку, в которую была произведена запись. «Разделяемая» линия при кэш-промахе чтения служит для информирования контроллера локальной кэш-памяти о месте, откуда поступила копия строки: из основной памяти или другого кэша. Таким образом, состояние S применяется только к тем данным, которые действительно используются совместно [162, 212].

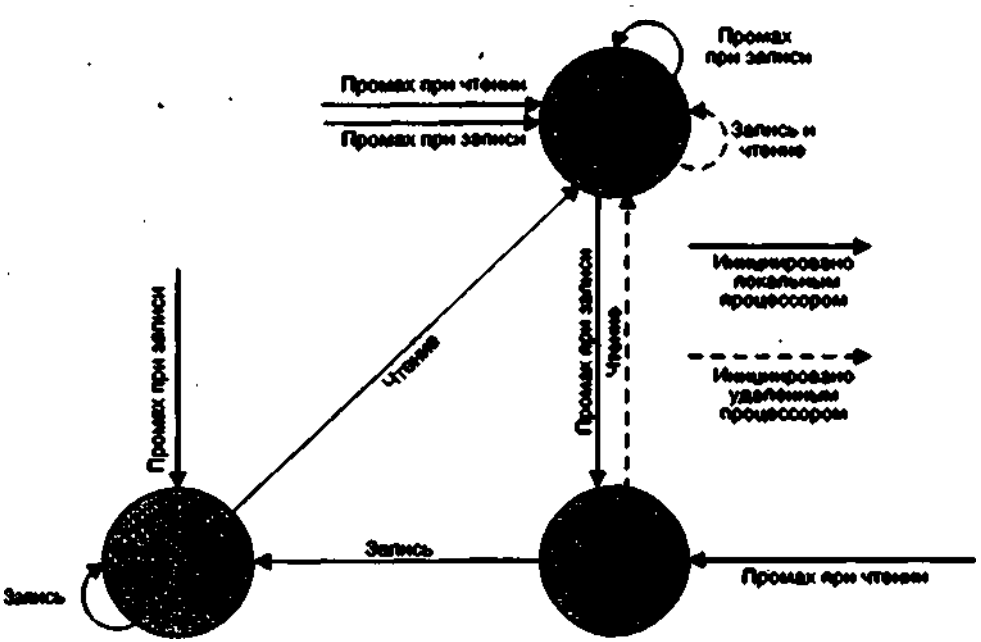


Рис. 11.12. Протокол Firefly

Протокол имеет преимущества перед ранее описанными в том, что стратегия сквозной записи привлекается лишь по логической необходимости. Когда ячейка перестает быть совместно используемой, нужна только одна дополнительная операция записи, которая производится последней кэш-памятью, содержащей эту ячейку. Это приводит к тому, что протокол Firefly существенно экономнее по трафику шины по сравнению с прочими протоколами [211]. С другой стороны, стратегия сквозной записи остается в силе до тех пор, пока строка кэша будет совместно используемой, даже если фактически чтение строки и запись в нее производит только один процессор. Отсюда — увеличение трафика шины, что доказывает неперспективность использования данного протокола в будущих разработках протоколов обеспечения когерентности кэш-памяти.

Протокол Dragon. Протокол применен в мультипроцессорной системе Xerox Dragon и представляет собой независимую версию протокола Firefly.

В протоколе реализована процедура записи с обновлением. Строка кэша может иметь одно из пяти состояний [162,212]:

- *Invalid (I)* - копия, хранящаяся в кэше, недействительна;
- *Read Private (RP)* — существует лишь одна копия блока, и она совпадает с содержимым основной памяти;
- *Private Dirty* (PD) — существует лишь одна копия блока, и она не совпадает с содержимым основной памяти;
- *Shared Clean* (SC) — имеется несколько копий блока, и все они идентичны содержимому основной памяти;
- *Shared Dirty* (SO) — имеется несколько копий блока, не совпадающих с содержимым основной памяти.

Дополнительное состояние SO предназначено для предотвращения записи в основную память. Диаграмма состояний для данного протокола приведена на рис. 11.13.

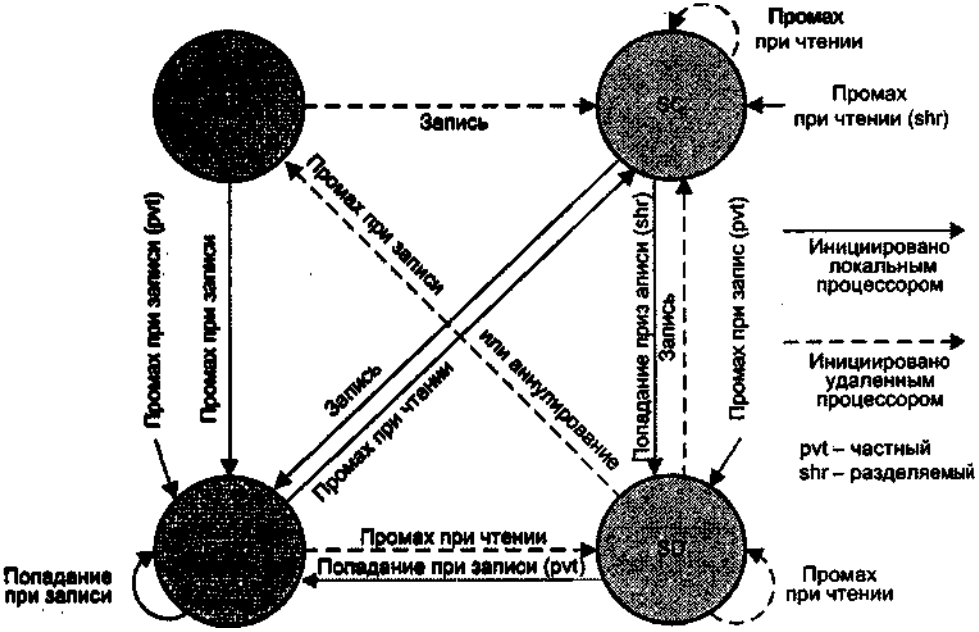


Рис. 11.13. Протокол Dragon

Протокол **MESI.** Безусловно, среди известных протоколов наблюдения самым популярным является протокол MESI (Modified/Exclusive/Shared/Invalid). Протокол MESI широко распространен в коммерческих микропроцессорных системах, например на базе микропроцессоров Pentium и PowerPC. Так, его можно обнаружить во внутреннем кэше и контроллере внешнего кэша i82490 микропроцессора Pentium, в процессоре i860 и контроллере кэш-памяти MC88200 фирмы Motorola,

Протокол был разработан для кэш-памяти с обратной записью. Одной из основных задач протокола MESI является откладывание на максимально возмож-

ный срок операции обратной записи кэшированных данных в основную память ВС. Это позволяет улучшить производительность системы за счет минимизации ненужных пересылок информации между кэшами и основной памятью.

Протокол MESI приписывает каждой кэш-строке одно из четырех состояний, которые контролируются двумя битами состояния MESI в теге данной строки. Статус кэш-строки может быть изменен как процессором, для которого эта кэш-память является локальной, так и другими процессорами мультипроцессорной системы. Управление состоянием кэш-строк может быть возложено и на внешние логические устройства. Одна из версий протокола предусматривает использование ранее рассмотренной схемы однократной записи.

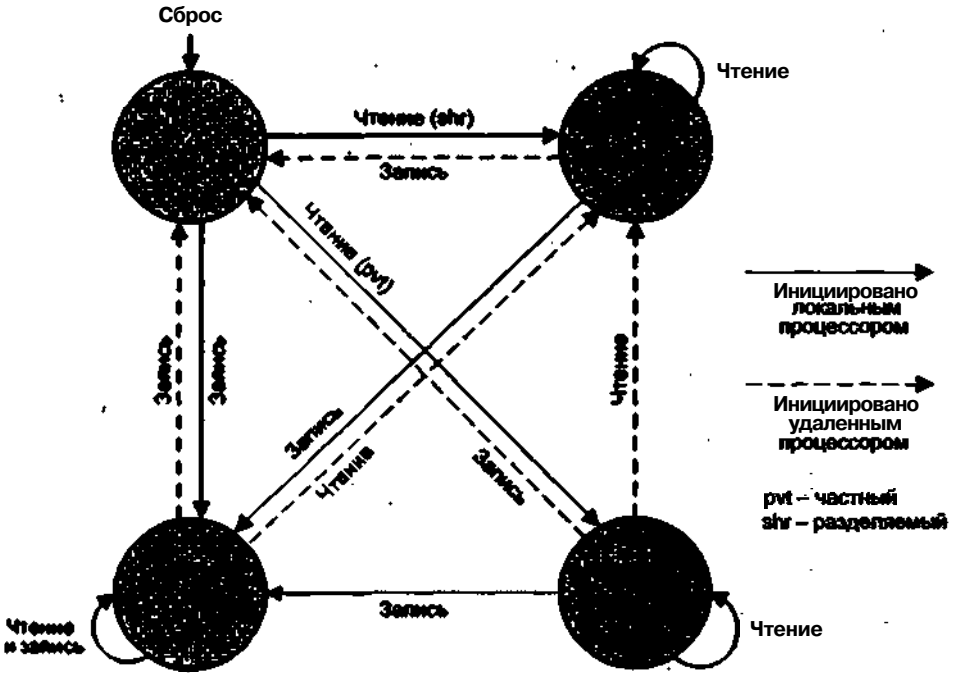


Рис. 11.14, Протокол MESI -диаграмма состояний без учета однократной записи

Согласно протоколу MESI, каждая строка бывает в одном из четырех возможных состояний (в дальнейшем будем ссылаться на эти состояния с помощью букв M, E, S и I):

- *Модифицированная* (M, Modified) - данные в кэш-строке, помеченной как M, были модифицированы, но измененная информация пока не переписана в основную память. Это означает, что информация, содержащаяся в рассматриваемой строке, достоверна только в данном кэше, а в основной памяти и остальных кэшах — недостоверна.
- *Эксклюзивная* (E, Exclusive) — данная строка в кэше не подвергалась изменению посредством запроса на запись, совпадает с аналогичной строкой в основной памяти, но отсутствует в любом другом локальном кэше. Иными словами, она достоверна в этом кэше и недостоверна в любом другом.

- *Разделяемая* (S, Shared) - строка в кэше совпадает с аналогичной строкой в основной памяти (данные достоверны) и может присутствовать в одном или нескольких из прочих кэшей.
- *Недействительная* (I, Invalid) - кэш-строка, помеченная как недействительная, не содержит достоверных данных и становится логически недоступной.

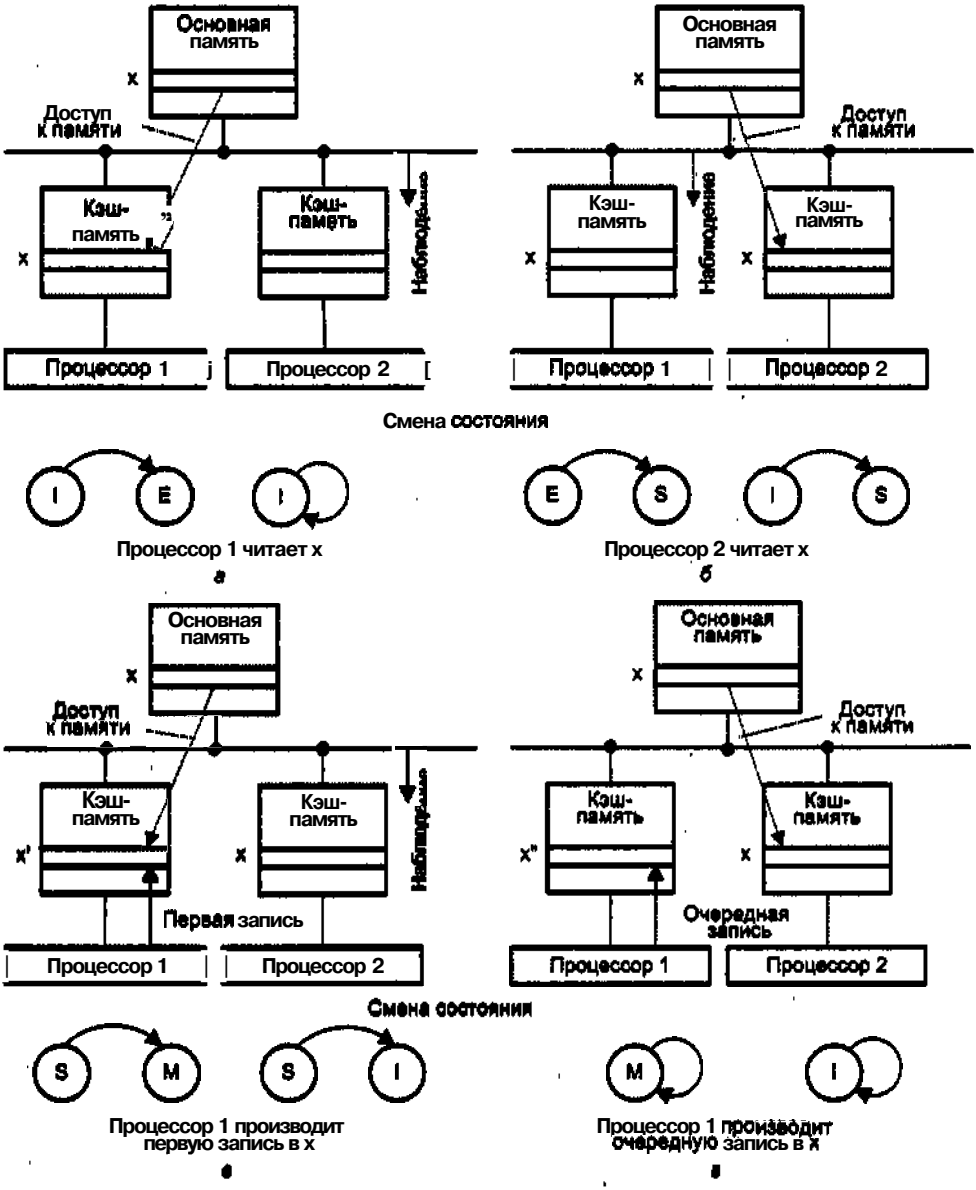


Рис. 11.15. Последовательность смены состояний в протоколе MESI: *а* — процессор 1 читает x; *в* — процессор 2 читает x; *г* — процессор 1 производит первую запись в x; *д* — процессор 1 производит очередную запись в x

Порядок перехода строки кэш-памяти из одного состояния в другое зависит от: текущего статуса строки, выполняемой операции (чтение или запись), результата обращения в кэш (попадание или промах) и, наконец, от того, является ли строка совместно используемой или нет. На рис. 11.14 приведена диаграмма основных переходов без учета режима однократной записи.

Предположим, что один из процессоров делает запрос на чтение из строки, которой в текущий момент нет в его локальной кэш-памяти (промах при чтении). Запрос будет широковещательно передан по шине. Если ни в одном из кэшей не нашлось копии нужной строки, то ответной реакции от контроллеров наблюдения других процессоров не последует, строка будет считана в кэш запросившего процессора из основной памяти, а копии будет присвоен статус E. Если в каком-либо из локальных кэшей имеется искомая копия, от соответствующего контроллера слежения поступит отклик, означающий доступ к совместно используемой строке. Все копии рассматриваемой строки во всех кэшах будут переведены в состояние S, вне зависимости от того, в каком состоянии они были до этого (M, E или S).

Когда процессор делает запрос на запись в строку, отсутствующую в его локальной кэш-памяти (промах при записи), перед загрузкой в кэш-память строка должна быть считана из основной памяти (ОП) и модифицирована. Прежде чем процессор сможет загрузить строку, он должен убедиться, что в основной памяти действительно находится достоверная версия данных, то есть что в других кэшах отсутствует модифицированная копия данной строки. Формируемая в этом случае последовательность операций носит название *чтения с намерением модификации* (RWITM, Read With Intent To Modify). Если в одном из кэшей обнаружилась копия нужной строки, причем в состоянии M, то процессор, обладающий этой копией, прерывает RWITM-последовательность и переписывает строку в ОП, после чего меняет состояние строки в своем кэше на I. Затем RWITM-последовательность возобновляется и делается повторное обращение к основной памяти для считывания обновленной строки. Окончательным состоянием строки будет M, при котором ни в ОП, ни в других кэшах нет еще одной достоверной ее копии. Если копия строки существовала в другом кэше и не имела состояния M, то такая копия аннулируется и доступ к основной памяти производится немедленно.

Кэш-попадание при чтении не изменяет статуса читаемой строки. Если процессор выполняет доступ для записи в существующую строку, находящуюся в состоянии S, он передает на шину широковещательный запрос, с тем чтобы информировать другие кэши, обновляет строку в своем кэше и присваивает ей статус M. Все остальные копии строки переводятся в состояние I. Если процессор производит доступ по записи в строку, находящуюся в состоянии E, единственное, что он должен сделать, — это произвести запись в строку и изменить ее состояние на M, поскольку другие копии строки в системе отсутствуют.

На рис. 11.15 показана типичная последовательность событий в системе из двух процессоров, запрашивающих доступ к ячейке x. Обращение к любой ячейке строки кэш-памяти рассматривается как доступ ко всей строке.

Проиллюстрируем этапы, когда процессор 2 пытается прочитать содержимое ячейки x" (рис. 11.16). Сперва наблюдается кэш-промах по чтению и процессор пытается обратиться к основной памяти. Процессор 1 следит за шиной, обнаруживает обращение к ячейке, копия которой есть в его кэш-памяти и находится в

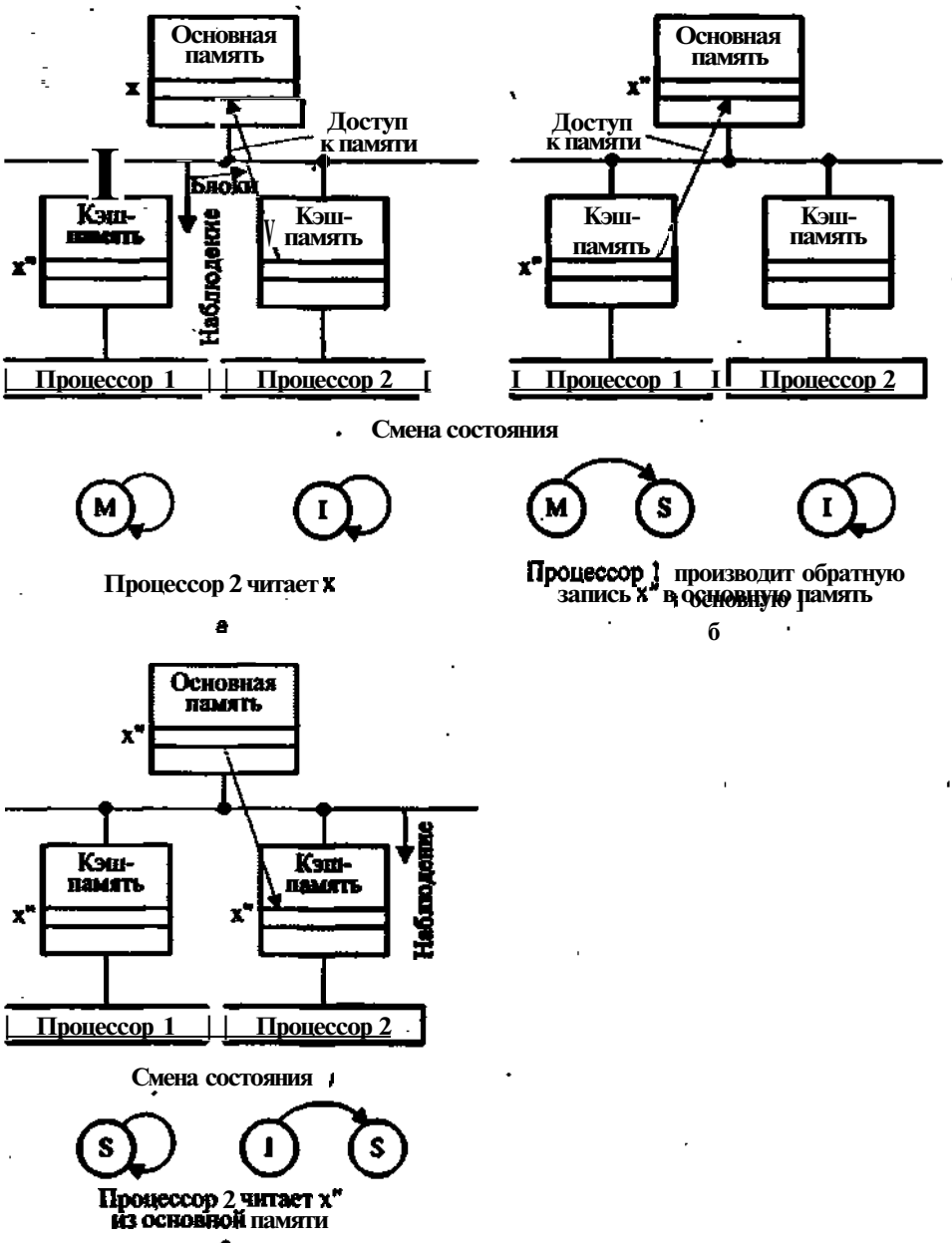


Рис 11.16. Переход из состояния E в состояние S в протоколе MESI: *а* — процессор 2 читает x; *б* — процессор1 производит обратную запись x'' в основную память: *в* — процессор2 читает x'' из основной памяти

состоянии M, поэтому он блокирует операцию чтения от процессора 2. Затем процессор 1 переписывает строку, содержащую x'', в ОП и освобождает процессор *2;* чтобы тот мог повторить доступ к основной памяти. Теперь процес-

сор 2 получает строку, содержащую x", и загружает ее в свою кэш-память. Обе копии помечаются как S.

До сих пор рассматривалась версия протокола MESI без однократной записи. С учетом однократной записи диаграмма состояний, изображенная на рис. 11.14, немного видоизменяется. Все кэш-промахи при чтении вызывают переход в состояние S. Первое попадание при записи сопровождается переходом в состояние E (так называемый переход однократной записи). Следующее попадание при записи влечет за собой изменение статуса строки на M.

Протоколы на основе справочника

Протоколы обеспечения когерентности на основе справочника характерны для сложных мультипроцессорных систем с совместно используемой памятью, где процессоры объединены многоступенчатой иерархической сетью межсоединений. Сложность топологии приводит к тому, что применение протоколов наблюдения с их механизмом широковещания становится дорогостоящим и неэффективным.

Протоколы на основе справочника предполагают сбор и отслеживание информации о содержимом всех локальных кэшей. Такие протоколы обычно реализуются с помощью централизованного контроллера, физически представляющего собой часть контроллера основной памяти. Собственно справочник хранится в основной памяти. Когда контроллер локальной кэш-памяти делает запрос, контроллер справочника обнаруживает такой запрос и формирует команды, необходимые для пересылки данных из основной памяти либо из другой локальной кэш-памяти, содержащей последнюю версию запрошенных данных. Центральный контроллер отвечает за обновление информации о состоянии локальных кэшей, поэтому он должен быть извещен о любом локальном действии, способном повлиять на состояние блока данных.

Справочник содержит множество записей, описывающих каждую кэшируемую ячейку ОП, которая может быть совместно использована процессорами системы. Обращение к справочнику производится всякий раз, когда один из процессоров изменяет копию такой ячейки в своей локальной памяти. В этом случае информация из справочника нужна для того, чтобы аннулировать или обновить копии измененной ячейки (или всей строки, содержащей эту ячейку) в прочих локальных кэшах, где такие копии имеются.

Для каждой строки общего пользования, копия которой может быть помещена в кэш-память, в справочнике выделяется одна запись, хранящая указатели на копии данной строки. Кроме того, в каждой записи выделен один бит модификации (D), показывающий, является ли копия «грязной»- (D = 1 - dirty) или «чистой»- (D = 0 - clean), то есть изменялось ли содержимое строки в кэш-памяти после того, как она была туда загружена. Этот бит указывает, имеет ли право процессор производить запись в данную строку.

В настоящее время известны три способа реализации протоколов обеспечения когерентности кэш-памяти на основе справочника: полный справочник, ограниченные справочники и сцепленные справочники.

В протоколе *полного справочника* единый централизованный справочник поддерживает информацию обо всех кэшах. Справочник хранится в основной памяти.

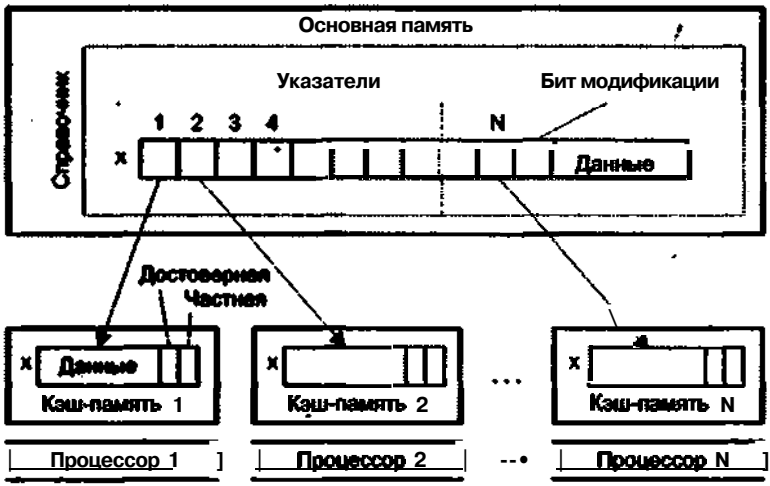


Рис. 11.17. Протокол обеспечения когерентности кэш-памяти с полным справочником

В системе из N процессоров каждая запись справочника будет содержать *N* однобитовых указателей. Если в соответствующей локальной кэш-памяти присутствует копия данных, бит-указатель устанавливается в 1, иначе - в 0. Схема с полным справочником показана на рис. 11.17. Здесь предполагается, что копия строки имеется в каждом кэше. Каждой строке придаются два индикатора состояния: бит достоверности (V, Valid) и бит владения (P, Private). Если информация в строке корректна, ее V-бит устанавливается в 1. Единичное значение P-бита указывает, что данному процессору предоставлено право на запись в соответствующую строку своей локальной кэш-памяти.

Предположим, что процессор 2 производит запись в ячейку x. В исходный момент процессор не получил еще разрешения на такую запись. Он формирует за- В ответ на запрос во все кэши, где есть копии строки, содержащей ячейку x, вы дается сигнал аннулирования имеющихся копий. Каждый кэш, получивший этот сигнал, сбрасывает бит достоверности аннулируемой строки (V-бит) в 0 и возвращает контроллеру справочника сигнал подтверждения. После приема всех сигналов подтверждения контроллер справочника устанавливает в единицу бит модификации (D-бит) соответствующей записи справочника и посылает процессору 2 сигнал, разрешающий запись в ячейку x. С этого момента процессор 2 может продолжить запись в собственную копию ячейки x, а также в основную память, если в кэше реализована схема сквозной записи.

Основные проблемы протокола полного справочника связаны с большим количеством записей. Для каждой ячейки в справочнике системы из N процессоров требуется N+ 1 бит, то есть с увеличением числа процессоров коэффициент сложности возрастает линейно. Протокол полного справочника допускает наличие в каждом локальном кэше копий всех совместно используемых ячеек. На практике такая возможность далеко не всегда остается востребованной - в каждый конкретный момент обычно актуальны лишь одна или несколько копий. В *протоколе с ограниченными справочниками* копии отдельной строки вправе находиться только

в ограниченном числе кэшей - одновременно может быть не более чем и копий строки, при этом число указателей в записях справочника уменьшается до n ($n < N$). Чтобы однозначно идентифицировать кэш-память, хранящую копию, указатель вместо одного бита должен состоять из $\log_2 N$ бит, а общая длина указателей в каждой записи справочника вместо N бит будет равна $n\log_2 N$ бит. При постоянном значении n темпы роста коэффициента сложности ограниченного справочника по мере увеличения размера системы ниже, чем в случае линейной зависимости.

Когда одновременно требуется более чем « копий, контроллер принимает решение, какие из копий сохранить, а какие аннулировать, после чего производятся соответствующие изменения в указателях записей справочника.

Метод *сцепленных справочников* также имеет целью сжать объем справочника. В нем для хранения записей привлекается связный список, который может быть реализован как односвязный (однонаправленный) и двусвязный (двунаправленный).

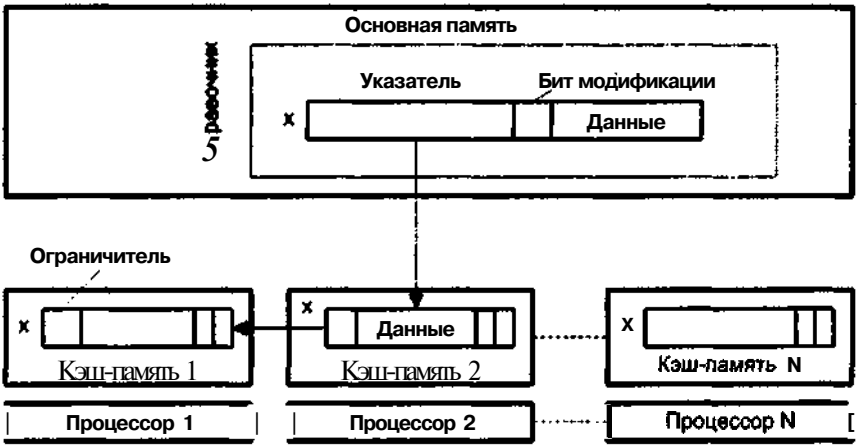


Рис. 11.18. Протокол обеспечения когерентности кэш-памяти со сцепленным справочником

В односвязном списке (рис. 11.18) каждая запись справочника содержит указатель на копию строки в одном из локальных кэшей. Копии одноименных строк в разных кэшах системы образуют однонаправленную цепочку. Для этого в их тегах предусмотрено специальное поле, куда заносится указатель на кэш-память, содержащую следующую копию цепочки. В тег последней копии цепочки помещается специальный символ-ограничитель. Сцепленный справочник допускает цепочки длиной в N, то есть поддерживает N копий ячейки. При создании еще одной копии цепочку нужно разрушить, а вместо нее сформировать новую. Пусть, например, в процессоре 5 нет копии ячейки x и он обращается за ней к основной памяти. Указатель в справочнике изменяется так, чтобы указывать на кэш с номером 5, а указатель в кэше 5 — таким образом, чтобы указывать на кэш *2.* Для этого контроллер основной памяти наряду с затребованными данными должен передать в кэш-память 5 также и указатель на кэш-память с номером 2. Лишь после того, как будет сформирована вся структура цепочки, процессор 5 получит разрешение на доступ к ячейке x. Если процессор производит запись в ячейку, то вниз по тракту, определяемому соответствующей цепочкой указателей, посылается сигнал аннулирова-

ния. Цепочка должна обновляться и при удалении копии из какой-либо кэш-памяти.

Двусвязный список поддерживает указатели как в прямом, так и в обратном направлениях. Это позволяет более эффективно вставлять в цепочку новые указатели или удалять из нее уже не нужные, но требует хранения большего числа указателей.

Схемы на основе справочника «страдают» от «заторов» в централизованном контроллере, а также от коммуникационных издержек в трактах между контроллерами локальных кэшей и центральным контроллером. Тем не менее они оказываются весьма эффективными в мультипроцессорных системах со сложной топологией взаимосвязей между процессорами, где невозможно реализовать протоколы наблюдения.

Ниже дана краткая характеристика актуальных на настоящее время протоколов обеспечения когерентности кэш-памяти на основе справочника. Для детального ознакомления с этими протоколами приведены ссылки на соответствующие литературные источники.

Протокол Tang. Здесь присутствует централизованный глобальный справочник, содержащий полную копию всей информации из каталогов каждого из локальных кэшей [212]. Это приводит к проблеме узких мест, а также требует поиска соответствующих входов.

Протокол Censier. В схеме справочника Censier для указания того, какие процессоры содержат локальную копию данного блока памяти, используется битовый вектор указателей. Такой вектор имеется для каждого блока памяти. Недостатками метода является его неэффективность при большом числе процессоров, и, кроме того, для обновления строк кэша требуется доступ к основной памяти [155].

Протокол Archibald. Схема справочника Archibald — это пара замысловатых схем для иерархически организованных сетей процессоров, С детальным описанием этого протокола можно ознакомиться в [52].

Протокол Stenstrom. Справочник Stenstrom для каждого блока данных предусматривает шесть допустимых состояний. Этот протокол относительно прост и подходит для любых топологий межсоединений процессоров. Справочник хранится в основной памяти. В случае кэш-промаха при чтении происходит обращение к основной памяти, которая посылает сообщение кэш-памяти, являющейся владельцем блока, если такой находится. Получив это сообщение, кэш-владелец посылает затребованные данные, а также направляет сообщение всем остальным процессорам, совместно использующим эти данные, для того чтобы они обновили свои битовые векторы. Схема не очень эффективна при большом числе процессоров, однако в настоящее время это наиболее проработанный и широко распространенный протокол на основе справочника [155].

Контрольные вопросы

1. Проанализируйте влияние особенностей ВС с общей памятью и ВС с распределенной памятью на разработку программного обеспечения. Почему эти ВС называют соответственно сильно связанными и слабо связанными?
2. Поясните идею с чередованием адресов памяти. Из каких соображений выбирается механизм распределения адресов? Как он связан с классом архитектуры ВС?

3. Дайте сравнительную характеристику однородного и неоднородного доступов к памяти.
4. В чем заключаются преимущества архитектуры COMA?
5. Проведите сравнительный анализ моделей с кэш-когерентным и кэш-некогерентным доступом к неоднородной памяти.
6. Сформулируйте достоинства и недостатки архитектуры без прямого доступа к удаленной памяти.
7. Объясните смысл распределенной и совместно используемой памяти.
8. Разработайте свой пример, иллюстрирующий проблему когерентности кэш-памяти.
9. Охарактеризуйте особенности программных способов решения проблемы когерентности, выделите их преимущества и слабые стороны.
10. Сравните методики записи в память с аннулированием и записи в память с трансляцией, акцентируя их достоинства и недостатки.
11. Дайте сравнительную характеристику методов для поддержания когерентности в мультипроцессорных системах.
12. Выполните сравнительный анализ известных вам протоколов наблюдения.
13. Какой из протоколов наблюдения наиболее популярен? Обоснуйте причины повышенного к нему интереса.
14. Дайте развернутую характеристику протоколов когерентности на основе справочника и способов их реализации. В чем суть отличий этих протоколов от протоколов наблюдения?

Глава 12
Топологии вычислительных систем

В основе архитектуры любой многопроцессорной вычислительной системы лежит способность к обмену данными между компонентами этой ВС. Коммуникационная система ВС представляет собой *сеть, узлы* которой связаны трактами передачи данных — *каналами*. В роли узлов могут выступать процессоры, модули памяти, устройства ввода/вывода, коммутаторы либо несколько перечисленных элементов, объединенных в группу. Организация внутренних коммуникаций вычислительной системы называется *топологией*.

Топологию сети межсоединений (СМС) определяет множество узлов N объединенных множеством каналов С. Связь между узлами обычно реализуется по *двухточечной схеме* (point-to-point). Любые два узла, связанные каналом связи, называют смежными узлами или соседями. Каждый канал $c = (x, y) \in C$ соединяет один *узел-источник* (source node) *x* с одн и м *узлом-получателем* (recipient node) *y*, где $x, y \in N$. Узел-источник, служащий началом канала с, будем обозначать s_c а узел-получатель — второй конец канала — r_c. Часто пары узлов соединяют два канала - по одному в каждом направлении. Канал $c = (x, y)$ характеризуется шириной (w_c или w_{xy}) - числом сигнальных линий; частотой (f_c или f_{xy}) - скоростью передачи битов по каждой сигнальной линии; задержкой (t_c или t_{xy}) - временем пересылки бита из узла *x* в узел *y*. Для большинства каналов задержка находится в прямой зависимости от физической длины линии связи (l_c) и скорости распространения сигнала (v): $l = vt$ Полоса пропускания канала b определяется выражением $b = wf$.

В зависимости от того, остается ли конфигурация взаимосвязей неизменной, по крайней мере пока выполняется определенное задание, различают сети со *статической* и *динамической* топологиями. В статических сетях структура взаимосвязей фиксирована. В сетях с динамической топологией в процессе вычислений конфигурация взаимосвязей с помощью программных средств может быть оперативно изменена.

Узел в сети может быть терминальным, то есть источником или приемником данных, коммутатором, пересылающим информацию с входного порта на выходной, или совмещать обе роли. В сетях *с непосредственными связями* (direct networks)

каждый узел одновременно является как терминальным узлом, так и коммутатором, и сообщения пересылаются между терминальными узлами напрямую. В *сетях с косвенными связями* (indirect networks) узел может быть либо терминальным, либо коммутатором, но не одновременно, поэтому сообщения передаются опосредовано, с помощью выделенных коммутирующих узлов. (В дальнейшем для простоты изложения позволим называть оба варианта «прямыми» и «косвенными» сетями, также для краткости вместо терминального узла будем говорить «терминал», несмотря на некоторую языковую некорректность.) Существуют также такие топологии, которые нельзя однозначно причислить ни к прямым, ни к косвенным. Любую прямую СМС можно изобразить в виде косвенной, разделив каждый узел на два — терминальный узел и узел коммутации.. Современные прямые сети реализуются именно таким образом — коммутатор отделяется от терминального узла и помещается в выделенный маршрутизатор. Основное преимущество прямых СМС в том, что коммутатор может использовать ресурсы терминальной части своего узла. Это становится существенным, если учесть, что, как правило, последний включает в себя вычислительную машину или процессор.

Тремя важнейшими атрибутами СМС являются:

- стратегия синхронизации;
- стратегия коммутации;
- стратегия управления.

Две возможных стратегии синхронизации операций в сети — это *синхронная* и *асинхронная*. В синхронных СМС все действия жестко согласованы во времени, что обеспечивается за счет единого генератора тактовых импульсов (ГТИ), сигналы которого одновременно транслируются во все узлы. В асинхронных сетях единого генератора нет, а функции синхронизации распределены по всей системе, причем в разных частях сети часто используются локальные ГТИ.

В зависимости от выбранной стратегии коммутации различают *сети с коммутацией соединений и сети с коммутацией пакетов. Как в первом, так и во втором* варианте информация пересылается в виде пакета. *Пакет* представляет собой группу битов, для обозначения которой применяют также термин *сообщение*.

В сетях с коммутацией соединений путем соответствующей установки коммутирующих элементов сети формируется тракт от узла-источника до узла-получателя, сохраняющийся, пока весь доставляемый пакет ни достигнет пункта назначения. Пересылка сообщений между определенной парой узлов производится всегда по одному и тому же маршруту.

Сети с коммутацией пакетов предполагают, что сообщение самостоятельно находит свой путь к месту назначения. В отличие от сетей с коммутацией соединений, маршрут от исходного пункта к пункту назначения каждый раз может быть иным. Пакет последовательно проходит через узлы сети. Очередной узел запоминает принятый пакет в своем буфере временного хранения, анализирует его и делает выводы, что с ним делать дальше. В зависимости от загруженности сети принимается решение о возможности немедленной пересылки пакета к следующему узлу и о дальнейшем маршруте следования пакета на пути к цели. Если все возможные тракты для перемещения пакета к очередному узлу заняты, в буфере узла

формируется очередь пакетов, которая «рассасывается» по мере освобождения линий связи между узлами (если очередь также насыщается, согласно одной из стратегий маршрутизации может произойти так называемый «сброс хвоста» (tail drop), отказ от вновь поступающих пакетов).

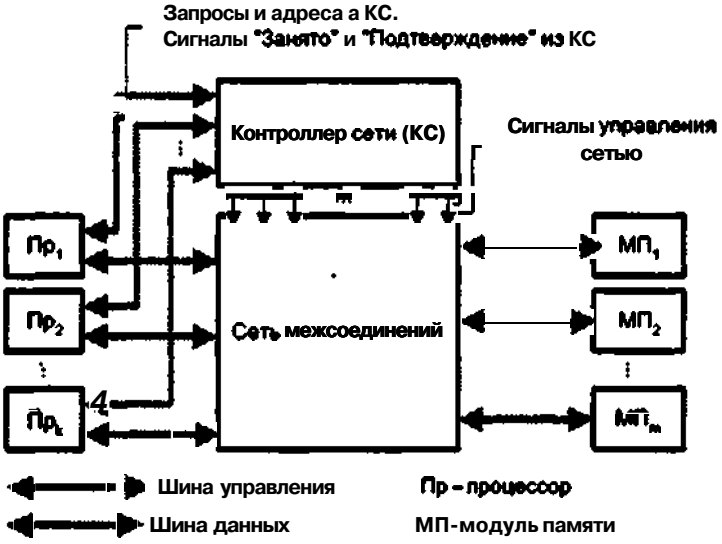


Рис. 12.1. Структура сети с централизованным управлением

СМС можно также классифицировать по тому, как в них организовано управление. В некоторых сетях, особенно с переключением соединений, принято *централизованное управление* (рис. 12.1). Процессоры посылают запрос на обслуживание в единый контроллер сети, который производит арбитраж запросов с учетом заданных приоритетов и устанавливает нужный маршрут. К данному типу следует отнести сети с шинной топологией. Процессорные матрицы также строятся как сети с централизованным управлением, которое осуществляется сигналами от центрального процессора. Приведенная схема применима и к сетям с коммутацией пакетов. Здесь тег маршрутизации, хранящийся в заголовке пакета, определяет адрес узла назначения. Большинство из серийно выпускаемых ВС имеют именно этот тип управления.

В схемах с *децентрализованным управлением* функции управления распределены по узлам сети.

Вариант с централизацией проще реализуется, но расширение сети в этом случае связано со значительными трудностями. Децентрализованные сети в плане подключения дополнительных узлов значительно гибче, однако взаимодействие узлов в таких сетях существенно сложнее.

В ряде сетей связь между узлами обеспечивается посредством множества коммутаторов, но существуют также сети с одним коммутатором. Наличие большого числа коммутаторов ведет к увеличению времени передачи сообщения, но позволяет использовать простые переключающие элементы. Подобные сети обычно строятся как многоступенчатые.

Метрики сетевых соединений

При описании СМС их обычно характеризуют с помощью следующих параметров:

- размера сети *(N);*
- числа связей (/);
- диаметра (D);
- порядка узла *(d);*
- пропускной способности (W);
- задержки (T);
- связности (Q);
- ширины бисекции (B);
- полосы бисекции (6).

Размер сети (network size) численно равен количеству узлов, объединяемых сетью.

Число связей (number of links) — это суммарное количество каналов между всеми узлами сети. В плане стоимости лучшей следует признать ту сеть, которая требует меньшего числа связей.

Диаметр сети (network diameter), называемый также *коммуникационным расстоянием,* определяет минимальный путь, по которому проходит сообщение между двумя наиболее удаленными друг от друга узлами сети. *Путь* (path) в сети — это упорядоченное множество каналов $P=\{c,, c_2,..., c_n\}$ *по* которым данные от узла-источника, последовательно переходя от одного промежуточного узла к другому, поступают на узел-получатель. Для обозначения отрезка пути между парой смежных узлов применяют термин *переход* (hop — в живой речи также «транзит» и «хоп»). Минимальный путь от узла *x* до узла *y* — это путь с минимальным числом переходов. Если обозначить число переходов в минимальном пути от узла *x* до узла *y* через *H(x, y),* то диаметр сети *D* — это наибольшее значение H *(x, y)* среди всех возможных комбинаций *x и y.* Так, в цепочке из четырех узлов наибольшее число переходов будет между крайними узлами, и «диаметр» такой цепочки равен трем. С возрастанием диаметра сети увеличивается общее время прохождения сообщения, поэтому разработчики ВС стремятся по возможности обходиться меньшим диаметром.

Порядок узла. Каждый узел сети связан с прочими узлами множеством каналов $C_{ix} = C_{ix} \cup C_{Ox}$, где $C_{ix} = \{c : r_c = x\}$ — множество входных каналов, а $C_{Ox} = \{c : s_c = x\}$ — множество выходных каналов. Порядок узла *x* представляет собой сумму числа входных $|C_{ix}|$ и выходных $|C_{Ox}|$ каналов узла, то есть равен числу узлов сети, с которыми данный узел связан напрямую. Например, в сети, организованной в виде матрицы, где каждый узел имеет каналы только к ближайшим соседям (слева, справа, сверху и снизу), порядок узла равен четырем. В вычислительной системе СМ-1, построенной по топологии гиперкуба, каждый узел связан с 12 другими узлами, следовательно, порядок узлов равен 12. Увеличение значения этой метрики ведет к усложнению коммутационных устройств сети и, как следствие, к дополнительным задержкам в передаче сообщений. С другой стороны, повышение порядка

узлов позволяет реализовать топологии, имеющие меньший диаметр сети, и тем самым сократить время прохождения сообщения. Так, если сеть, состоящая из 4096 узлов (2^{12} или 64^2), построена по матричной схеме, ее диаметр составляет 126, а для гиперкуба - только 12. Разработчики ВС обычно отдают предпочтение таким топологиям, где порядок всех узлов одинаков, что позволяет строить сети по модульному принципу. *

Пропускная способность сети (network bandwidth) характеризуется количеством информации, которое может быть передано по сети в единицу времени. Обычно измеряется в мегабайтах в секунду или гигабайтах в секунду без учета издержек на передачу избыточной информации, например битов паритета.

Задержка сети (network latency) - это время, требуемое на прохождение сообщения через сеть. В сетях, где время передачи сообщений зависит от маршрута, говорят о средней, минимальной и максимальной задержках сети.

Связность сети (network connectivity) можно определить как минимальное число узлов или линий связи, которые должны выйти из строя, чтобы сеть распалась на две непересекающихся сети. Следовательно, связность сети характеризует устойчивость сети к повреждениям, то есть ее способность обеспечивать функционирование ВС при отказе компонентов сети.

Ширина бисекции *(*bisection width*). Для начала* определим *понятие среза сети* (cut of network). Срез сети C(N1, N_2) — это множество каналов, разрыв которых разделяет множество узлов сети N на два непересекающихся набора узлов N_1 и N_2. Каждый элемент $C(N_1, N_2)$- это канал, соединяющий узел из набора N_1 с узлом из N_2 Бисекция сети - это срез сети, разделяющий ее примерно пополам, то есть так, что $|N_2| <= |N_1| <= |N_2| + 1$. Ширину бисекции B характеризуют минимальным числом каналов, разрываемых при всех возможных бисекциях сети:

$$B = \min_{bisection} |C(N_1, N_2)|.$$

Ширина бисекции позволяет оценить число сообщений, которые могут быть переданы по сети одновременно, при условии что это не вызовет конфликтов из-за попытки использования одних и тех же узлов или линий связи.

Полоса бисекции (bisection bandwidth) - это наименьшая полоса пропускания по всем возможным бисекциям сети. Она характеризует пропускную способность тех линий связи, которые разрываются при бисекции сети, и позволяет оценить наихудшую пропускную способность сети при попытке одномоментной передачи нескольких сообщений, если эти сообщения должны проходить из одной половины сети в другую. Полоса бисекции b определяется выражением $b = \min_{bisection} B(N_1, N_2)$ Для сетей с одинаковой шириной полосы во всех b_c каналах справедливо соотношение: b - b_c x B Малое значение полосы бисекции свидетельствует о возможности конфликтов при одновременной пересылке нескольких сообщений.

Функции маршрутизации данных

Кардинальным вопросом при выборе топологии СМС является способ маршрутизации данных, то есть правило выбора очередного узла, которому пересылается сообщение. Основой маршрутизации служат адреса узлов. Каждому узлу в сети

присваивается уникальный адрес. Исходя из этих адресов, а точнее, их двоичных представлений, производится соединение узлов в статических топологиях или их коммутация в топологиях динамических. В сущности, принятая система соответствия между двоичными кодами адресов смежных узлов — *функция маршрутизации данных* — и определяет сетевую топологию. Последнюю можно описать как набор функций маршрутизации, задающий порядок выбора промежуточных узлов на пути от узла-источника к узлу-получателю. В некоторых топологиях используется единая для всей СМС функция маршрутизации, в других — многоступенчатых — при переходе от одной ступени к другой может применяться иная функция маршрутизации.

Функция маршрутизации данных задает правило вычисления возможного адреса одного из смежных узлов по адресу второго узла. Сводится это к описанию алгоритма манипуляции битами адреса-источника для определения адреса-получателя. Ниже приводится формальное описание основных функций маршрутизации данных, применяемых в известных топологиях СМС, без анализа их достоинств и недостатков. Последнее, по мере надобности, будет сделано при рассмотрении конкретных топологий СМС. Для всех функций предполагается, что размерность сети равна N, а разрядность адреса — m, где $m = \log_2 N$. Биты адреса обозначены как b_i.

Перестановка

Функция перестановки (exchange) отвечает следующему соотношению:

$$E_k(b_m \ldots b_k, \ldots, b_1) = (b_m, \ldots, \overline{b_k} \ldots b_1), \quad 1 \le k \le m.$$

Работающим примером для данной функции маршрутизации может служить топология гиперкуба (рис. 12.2).

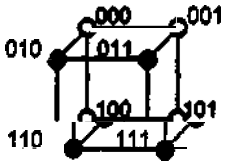


Рис. 12.2. Пример топологии с функцией перестановки (трехмерный гиперкуб)

Тасование

Функция тасования (shuffle) может быть реализована в одном из четырех вариантов: идеальное тасование (perfect shuffle), отсутствие тасования (unshuffle), субтасование по i-му биту (i^{th} subshuffle) и супертасование по i-му биту (i^{th} supershuffle). Ниже приведены формальные описания каждого из перечисленных вариантов, а на рис. 12.3 — примеры соответствующих им топологий.

- *идеальное тасование:*

$$S(b_m, b_{m-1}, \ldots, b_1) - (b_{m-1}, b_{m-2}, \ldots b_1, b_m).$$

Из приведенной формулы видно, что два узла с адресами *i и j* имеют между собой непосредственную связь при условии, что двоичный код j может быть

получен из двоичного кода i циклическим сдвигом влево. Идеальное Тасование - наиболее распространенный среди рассматриваемых вариантов функции тасования.

- *отсутствие тасования:* $U(b_m, b_{m-1}, ..., b_1) = (b_1, b_m, ..., b_2)$;
- *субтасование по i-му биту:* $S_i(b_m, b_{m-1}, ..., b_i, ..., b_1) = (b_m, ..., b_{i+1}, b_{i-1}, ..., b_1)$;
- *супертасование по i-му биту:* $S(b_m, b_{m-1}, ..., b_i, ..., b_1) = (b_{m-i}, ..., b_{m-i+1}, b_m, b_{m-1}, ..., b_1)$.

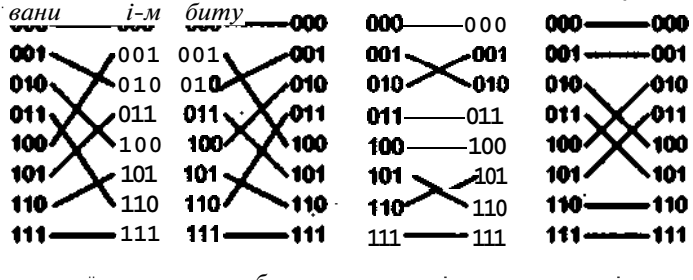


Рис. 12.3. Примеры топологий с тасованием для m = 3: *а* - идеальное тасование; *б* - отсутствие тасования; *в* - субтасование по второму биту; г - супертасование по второму биту

Баттерфляй

Функция «баттерфляй» (butterfly) - «бабочка» была разработана в конце 60-х годов Рабинером и Гоулдом. Свое название она получила из-за того, что построенная в соответствии с ней сеть по конфигурации напоминает крылья бабочки (рис. 12.4).

Математически функция может быть записана в виде

$$B(b_m, b_{m-1}, ..., b_1) = (b_1, b_{m-1}, ..., b_2, b_m).$$

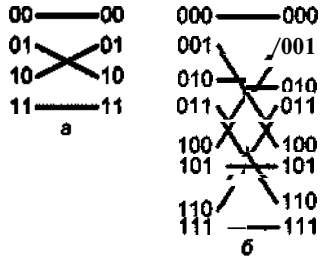


Рис. 12.4. Примеры топологии «баттерфляй» для: а - m = 2;б - m = 3

ХОТЯ функция -«баттерфляй» используется в основном при объединении ступеней в сетях.с динамической многоступенчатой топологией, известны также и -«чистые» «баттерфляй»-сети.

Реверсирование битов

Как следует из названия, функция сводится к перестановке битов адреса в обратном порядке:

$$R(b_m b_{m-1}, \ldots, b_1) = (b_1, b_2, \ldots, b_m).$$

Соответствующая топология для $m = 3$ показана на рис. 12.5. Хотя для значений $m \leq 3$ топология реверсирования битов совпадает с топологией «баттерфляй», при больших значениях m различия становятся очевидными.

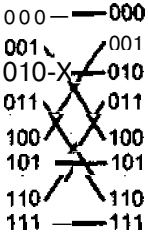


Рис. 12.5. Пример топологии на основе формулы реверсирования битов (m = 3)

Сдвиг

Функция маршрутизации по алгоритму сдвига имеет вид:

$$SH(x) - (x + 1) \bmod N \quad (N - 2").$$

При $m = 3$ данной функции соответствует топология кольца (рис. 12.6).

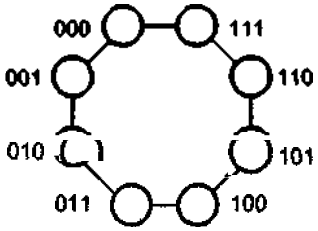


Рис. 12.6. Пример топологии на основе функции сдвига (m = 3)

Сеть ILLIACIV

Комбинируя несколько вариантов функции сдвига, можно образовать более сложные функции маршрутизации. Наиболее известной из таких «сложных» функций является *сеть ILLIAC IV,* впервые реализованная в топологии вычислительной системы ILLIAC IV:

$$R_{+1} = (i + 1) \bmod N;$$

$$R_{-1} = (i - 1) \bmod N;$$

$$R_{+r} = (i + r) \bmod N \, (0 \leq i \leq N - 1);$$

$$R_{-r} = 0' - r) \bmod N \quad (r = \sqrt{N}).$$

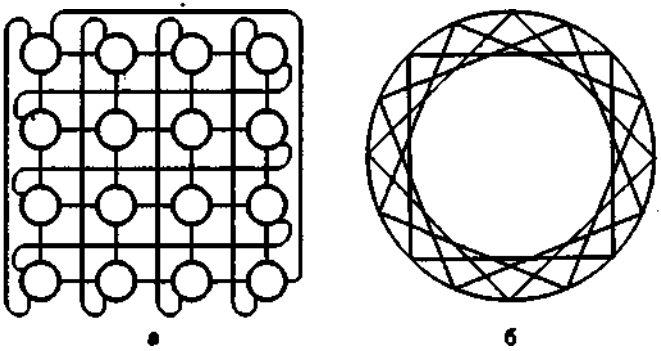


Рис. 12.7. Топологии сети ILUACIV: *а - в* виде решетки; *б - в* виде хордального кольца

Приведенные соотношения для N=4 отвечают двум вариантам топологии, показанным на рис. 12.7.

Первый вариант представляет собой фигуру, построенную на базе плоской решетки, где узлы в каждом столбце замкнуты в кольцо, а узлы в последовательных рядах соединены в замкнутую спираль. Второй вариант соответствует хордальному кольцу с шагом хорды, равным 4.

Циклический сдвиг

Функция циклического сдвига (barrel shift) описывается выражениями

$$B_{+i}(j) = (j + 2^i) \bmod N,$$

$$B_{-i}(j) = (j - 2^i) \bmod N, \text{ где } 0 \le j \le N - 1, 0 \le i \le \log_2 N - 1.$$

Топологию сети на базе рассматриваемой функции маршрутизации данных иллюстрирует рис. 12.8.

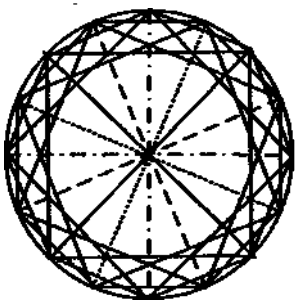


Рис. 12.8. Пример топологии на основе функции циклического сдвига ±2i

Статические топологии

К статическим топологиям СМС относят такие, где между двумя узлами возможен только один прямой фиксированный путь, то есть статические топологии не предполагают наличия в сети коммутирующих устройств. Если же такие устройства имеются, то используются они только перед выполнением конкретной за-

дачи, а в процессе всего времени вычислений топология СМС остается неизменной.

Из возможных критериев классификации статических сетей чаще всего выбирают их размерность. С этих позиций различают:

- одномерные топологии (линейный массив);
- двумерные топологии (кольцо, звезда, дерево, решетка, систолический массив);
- трехмерные топологии (полносвязная топология, хордальное кольцо);
- гиперкубическую топологию.

Ниже рассматриваются основные виды статических топологий СМС без акцентирования внимания на какой-либо их классификации, поскольку этот момент для поставленных в учебнике целей несущественен.

Линейная топология

В простейшей *линейной топологии* узлы сети образуют одномерный массив и соединены в *цепочки* (рис. 12.9). Линейная топология характеризуется следующими параметрами: $D = N - 1; d = 2; I = N - 1; B = 1.$

Рис. 12.9. Линейная топология

Линейная топология не обладает свойством полной симметричности, поскольку узлы на концах цепочки имеют только одну коммуникационную линию, то есть их порядок равен 1, в то время как порядок остальных узлов равен 2. Время пересылки сообщения зависит от расстояния между узлами, а отказ одного из них способен привести к невозможности пересылки сообщения. По этой причине в линейных СМС используют отказоустойчивые узлы, которые при отказе изолируют себя от сети, позволяя сообщению миновать неисправный узел.

Кольцевые топологии

Стандартная *кольцевая топология* представляет собой линейную цепочку, концы которой соединены между собой (рис, 12.10, *а)*. В зависимости от числа каналов между соседними узлами (один или два) различают *однонаправленные* и *двунаправленные* кольца. Кольцевая топология характеризуется следующими параметрами:

$$D = \min\left\lceil \frac{N}{2} \right\rceil \quad ; d = 2; I = N; B = 2.$$

Кольцевая топология, по сравнению с линейной, традиционно была менее популярной, поскольку добавление или удаление узла требует демонтажа сети.

Один из способов разрешения проблемы большого диаметра кольцевой сети - добавление линий связи в виде хорд, соединяющих определенные узлы кольца. Подобная топология носит название *хордальной*. Если хорды соединяют узлы с шагом 1 или N/2-1,диаметр сети уменьшается вдвое. На рис. 12.10,*б* показана *хордальная кольцевая сеть* с шагом 3.

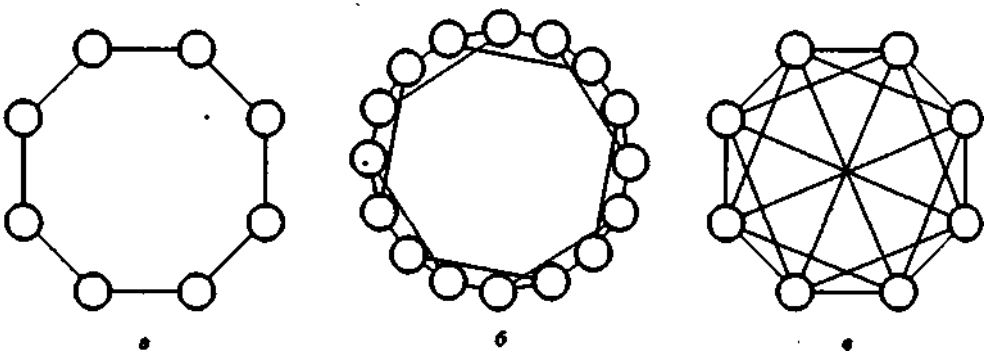


Рис. 12.10. Кольцевые топологии:а -стандартная;б-хордальная;в -с циклическим сдвигом связей

В качестве практических примеров для топологии кольца следует назвать сети Token Ring, разработанные фирмой IBM, а также вычислительные системы KSR1 и SCI.

Дальнейшее увеличение порядка узлов позволяет добиться еще большего сокращения тракта передачи сообщения. Примером такой топологии может служить показанная на рис. 12.10, *в* топология с *циклическим сдвигом связей*. Здесь стандартная кольцевая топология с *N* узлами дополнена соединениями между всеми узлами i и j, для которых $|i-j|$ совпадает с целой степенью числа 2. Алгоритмы маршрутизации для подобной сети чрезвычайно эффективны, однако порядок узлов по мере разрастания сети увеличивается.

Звездообразная топология

Звездообразная сеть объединяет множество узлов первого порядка посредством специализированного центрального узла - концентратора (рис. 12.11). Топология характеризуется такими параметрами: $D = 2; d = N1; I = N - 1; B \sim 1$.

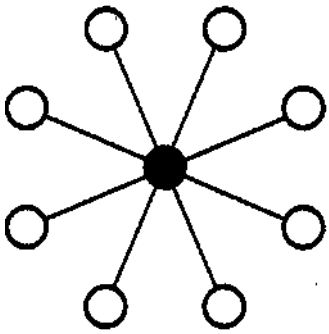


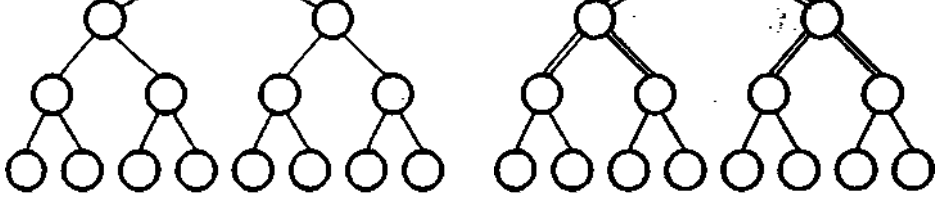
Рис. 12.11. Звездообразная топология

Звездообразная организация узлов и соединений редко используется для объединения процессоров многопроцессорной ВС, но хорошо работает, когда поток информации идет от нескольких вторичных узлов, соединенных с одним первичным узлом, например при подключении терминалов. Общая пропускная способ-

ность сети обычно ограничивается быстродействием концентратора, аналогично тому, как сдерживающим элементом в одношинной топологии выступает шина. По производительности эти топологии также идентичны. Основное преимущество звездообразной схемы в том, что конструктивное исполнение узлов на концах сети может быть очень простым.

Древовидные топологии

Еще одним вариантом структуры СМС является *древовидная топология* (рис. 12.12, *а)*. Сеть строится по схеме так называемого строго двоичного дерева, где каждый узел более высокого уровня связан с двумя узлами следующего по порядку более низкого уровня. Узел, находящийся на более высоком уровне, принято называть *родительским,* а два подключенных к нему нижерасположенных узла — *дочерними*. В свою очередь, каждый дочерний узел выступает в качестве родительского для двух узлов следующего более низкого уровня. Отметим, что каждый узел связан только с двумя дочерними и одним родительским. Если h — высота дерева (количество уровней в древовидной сети), определяемая как max [log2 N], то такую сетьможноохарактеризоватьследующимипараметрами:D=2(h- *1);d=3;l=N-* 1; *B = 1.* Так, вычислительная система из 262 144 узлов при решетчатой топологии (немного забегаем вперед) будет иметь диаметр 512, а в случае строго бинарного дерева — только 36. Топология двоичного дерева была использована в мультипроцессорной системе DADO из 1023 узлов, разработанной в Колумбийском университете.



а б

Рис. 12.12. Древовидная топология: *а* - стандартное дерево; *б* - "толстое" дерево

При больших объемах пересылок между несмежными узлами древовидная топология оказывается недостаточно эффективной, поскольку сообщения должны проходить через один или несколько промежуточных звеньев. Очевидно, что на более высоких уровнях сети вероятность затора из-за недостаточно высокой пропускной способности линий связи выше. Этот недостаток устраняют с помощью топологии, называемой *"толстым" деревом* (рис. 12.12, *б)*.

Идея "толстого" дерева состоит в увеличении пропускной способности коммуникационных линий на прикорневых уровнях сети. С этой целью на верхних уровнях сети родительские и дочерние узлы связывают не одним, а несколькими каналами, причем чем выше уровень, тем больше число каналов. На рисунке это отображено в виде множественных линий между узлами верхних уровней. Топология «толстого» дерева реализована в вычислительной системе СМ-5.

Решетчатые топологии

Поскольку значительная часть научно-технических задач связана с обработкой массивов, вполне естественным представляется стремление учесть это и в топологии ВС, ориентированных на подобные задачи. Такие топологии относят *к решетчатым* (mesh), а их конфигурация определяется видом и размерностью массива.

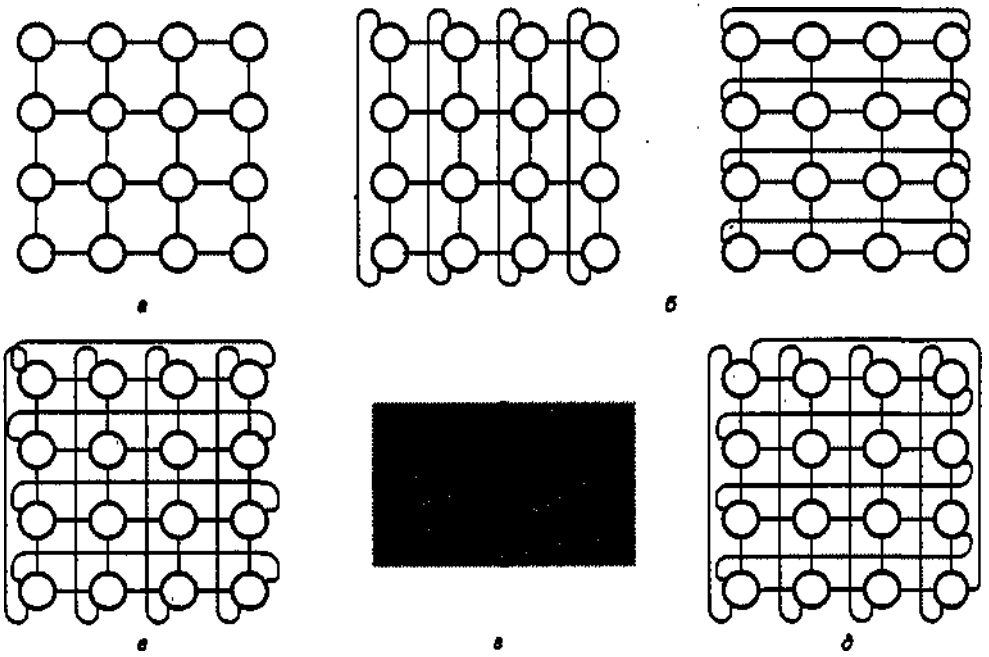


Рис. 12.13. Решетчатые топологии: а — **плоская;** б — цилиндрическая; **в–г — тороидальная;** **д** — витая тороидальная

Простейшими примерами для одномерных массивов могут служить цепочка и кольцо. Для двумерных массивов данных наиболее подходит топология плоской прямоугольной матрицы узлов, каждый из которых соединен с ближайшим соседом (рис. 12.13, а). Такая сеть размерности *mxm* $(m = \sqrt{N})$ имеет следующие характеристики; $D = 2(m-1); d=4; I=2N-2m; B-m$.

Если провести операцию *свертывания* (wraparound) плоской матрицы, соединив информационными трактами одноименные узлы левого и правого столбцов или одноименные узлы верхней и нижней строк плоской матрицы, то из плоской конструкции получаем топологию типа цилиндра (рис. 12.13,6*)*. В топологии цилиндра каждый ряд (или столбец) матрицы представляет собой кольцо. Если одновременно произвести свертывание плоской матрицы в обоих направлениях, получим тороидальную топологию сети (рис. 12.13, *в)*. Двумерный тор на базе решетки *m x m* обладает следующими параметрами:

$$D = 2\min\left[\frac{m}{2}\right]; d = 4; I = 2N; B = 2m.$$

Объемный вид тороидальной топологии для массива размерности 4x8 показан на рис. 12.13, г.

Помимо свертывания к плоской решетке может быть применена операция *скручивания* (twisting). Суть этой операции состоит в том, что вместо колец все узлы объединяются в разомкнутую или замкнутую спираль, то есть узлы, расположенные с противоположных краев плоской решетки, соединяются с некоторым сдвигом. Если горизонтальные петли объединены в виде спирали, образуется так называемая сеть типа ILLI AC. На рис. 12.13, *д* показана подобная конфигурация СМС, соответствующая хордальной сети четвертого порядка и характеризуемая следующими метриками: D = *m- 1; d = 4; / = 2N; B = 2m.*

Следует упомянуть и трехмерные сети. Один из вариантов, реализованный в архитектуре суперЭВМ Cray T3D, представляет собой трехмерный тор, образованный объединением процессоров в кольца по трем координатам: *x,y и z.*

Примерами ВС, где реализованы различные варианты решетчатых топологий, могут служить: ILLIAC IV, MPP, DAP, CM-2, Paragon и др.

Полносвязная топология

В *полносвязной топологии* (рис. 12.14), известной также под названием топологии *«максимальной гр у ппировки» или «топологии клика» (*clique - полный подграф [мат*ем.*]), каждый узел напрямую соединен со всеми остальными узлами сети. Сеть, состоящая из *N* узлов, имеет следующие параметры:

$$D = 1; d = N - 1; I = \frac{N(N-1)}{2}; B = \frac{N^2}{4}.$$

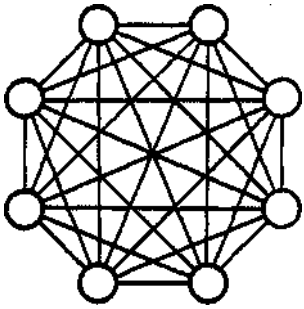


Рис. 12.14. Полносвязная топология

Если размер сети велик, топология становится дорогостоящей и трудно реализуемой. Более того, топология максимальной группировки не дает существенного улучшения производительности, поскольку каждая операция пересылки требует, чтобы узел проанализировал состояние всех своих *N*— 1 входов. Для ускорения этой операции необходимо, чтобы все входы анализировались параллельно, что, в свою очередь, усложняет конструкцию узлов.

Топология гиперкуба

При объединении параллельных процессоров весьма популярна топология гиперкуба, показанная на рис. 12.15. Линия, соединяющая два узла (рис. 12.15, *а).*

определяет одномерный гиперкуб. Квадрат, образованный четырьмя узлами (рис. 12.15, б) - двумерный гиперкуб, а куб из 8 узлов (рис. 12.15, *в)* - трехмерный гиперкуб и т. д. Из этого ряда следует алгоритм получения m-мерного гиперкуба: начинаем с (m - 1)-мерного гиперкуба, делаем его идентичную копию, а затем добавляем связи между каждым узлом исходного гиперкуба и одноименным узлом копии. Гиперкуб размерности *m (N* - 2^m) имеет следующие характеристики:

$$D = m; d = m; \frac{mN}{2}.$$

Увеличение размерности гиперкуба на 1 ведет к удвоению числа его узлов, увеличению порядка узлов и диаметра сети на единицу.

Обмен сообщениями в гиперкубе базируется на двоичном представлении номеров узлов. Нумерация узлов производится так, что для любой пары смежных узлов двоичное представление номеров этих узлов отличается только в одной позиции. В силу сказанного, узлы 0010 и 0110 — соседи, а узлы 0110 и 0101 таковыми не являются. Простейший способ нумерации узлов при создании m-мерного гиперкуба из двух *(m*- 1)-мерных показан на рис. 12.15, *д.* При копировании (m- 1)-мерного гиперкуба и соединении его с исходным *(m* - 1)-мерным гиперкубом необходимо, чтобы соединяемые узлы имели одинаковые номера. Далее к номерам узлов исходного гиперкуба слева добавляется бит, равный 0, а к номерам узлов копии - единичный бит,

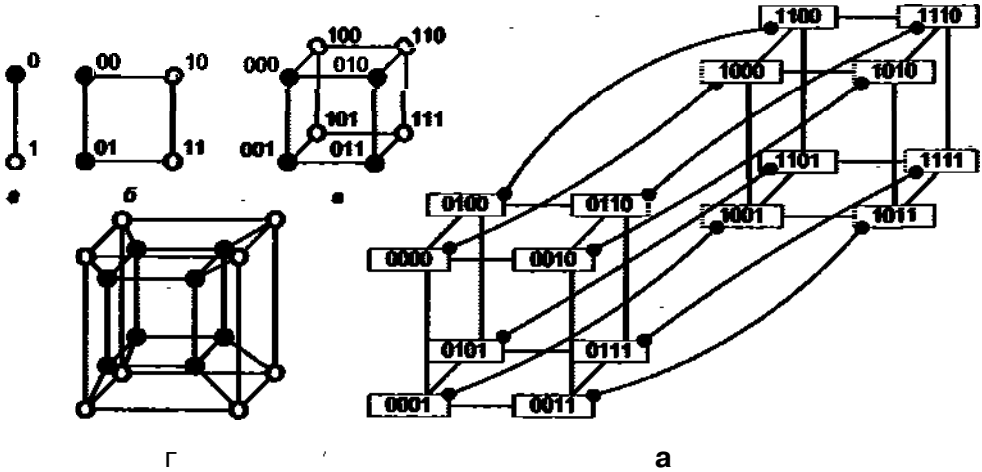


Рис. 12.15. Топология гиперкуба: *а* - одномерная; б -двумерная; в - трехмерная; *г* - четырехмерная; д - схема формирования четырехмерного гиперкуба из двух трехмерных

Номера узлов являются основой маршрутизации сообщений в гиперкубе. Такие номера в m-мерном гиперкубе состоят из *m* битов, а пересылка сообщения из узла *A* в узел *B* выполняется за m шагов. На каждом шаге узел может либо сохранить сообщение и не пересылать его дальше до следующего шага, либо отправить его дальше по одной из линий. На шаге *i* узел, хранящий сообщение, сравнивает (i-й бит своего собственного номера с i-м битом номера узла назначения. Если они совпадают, продвижение сообщения прекращается, если нет - сообщение переда-

ется вдоль линии i-го измерения. Линией i-го измерения считается та, которая была добавлена на этапе построения t-мерного гиперкуба из двух (i - 1)-мерных.

Создание гиперкуба при большом числе процессоров требует увеличения порядка узлов, что сопряжено с большими техническими проблемами. Компромиссное решение, несколько увеличивающее диаметр сети при сохранении базовой структуры, представляет собой *куб из циклически соединенных узлов* (рис. 12.16). Здесь порядок узла равен трем при любом размере сети.

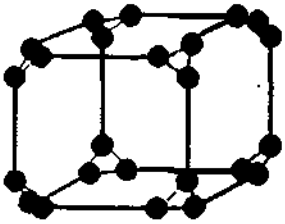


Рис. 12.16. Куб из циклически соединенных узлов третьего порядка

Топология k-ичного n-куба

Название топологии означает, что в ней реализуется куб, имеющий и измерений, причем каждое измерение содержит k узлов ($N = k^n$). Каждому узлу назначен n-разрядный номер в системе счисления с основанием ***к,*** и он связан с узлом, номер которого отличается только в одной цифре и только на единицу. k-ичный n-куб может быть построен путем объединения ***к*** экземпляров k-ичных (n - 1)-кубов в кольцо.

Многие ранее рассмотренные топологии представляют собой варианты топологии k-ичного n-куба:

- *k*-ичный 1-куб- кольцо;
- k-ичный 2-куб - двумерный тор;
- *k*-ичный 3-куб - трехмерный тор;
- 4-ичный 2-куб - плоская решетка 4x4;
- 2-ичный n-куб - гиперкуб.

Доказано, что эффективность топологии, а также ее масштабируемость улучшаются с ростом значения ***к*** и уменьшением количества измерений *n*.

Данные по рассмотренным статическим топологиям сведены в табл. 12.1.

Таблица 12.1. Характеристики сетей со статической топологией

Топология	Диаметр	Порядок узла	Число связей	Ширина бисекции	Симметричность	Размер сети
Полносвязная	1	$N-1$	$\frac{N(N-1)}{2}$	$\frac{N^2}{4}$	Да	N узлов
Звезда	2	1	$N-1$	1	Нет	*N узлов*
Двоичное дерево	2(A-1)	3	$N-1$	1	Нет	Высота дерева $h = \log_2 N$

продолжение

Таблица 12.1 *(продолжение)*

Топология	Диаметр	Порядок узла	Число связей	Ширина бисекции	Симметричность	Размер сети
Линейный массив	$N-1$	2	$N-1$	1	Нет	N узлов
Кольцо	$\left\lfloor \frac{N}{2} \right\rfloor$	2	N	2	Да	N узлов
Двумерная решетка	$2(m-1)$	4	$2N-2m$	$\sqrt{N}$	Нет	Решетка $m \times$ от, где $m = \sqrt{N}$
Двумерный тор	4fJ	4	$2N$	$2m$	Да	Тор $m \times m$, где $m = \sqrt{N}$
Гиперкуб	n	n	$\frac{nN}{2}$	$\frac{N}{2}$	Да	N узлов: $n = \log_2 N$
k-ичный n-куб	$n\left\lfloor \frac{k}{2} \right\rfloor$	2л	nN	$2k^{n-1}$	Да	$N = k^n$ узлов

Динамические топологии

Б *динамической топологии сети* соединение узлов обеспечивается электронными ключами, варьируя установки которых можно менять топологию сети. В отличие от ранее рассмотренных топологий, где роль узлов играют сами объекты информационного обмена, в узлах динамических сетей располагаются коммутирующие элементы, а устройства, обменивающиеся сообщениями (терминалы), подключаются к входам и выходам этой сети. В роли терминалов могут выступать процессоры или процессоры и модули памяти.

Если входы и выходы сети коммутирующих элементов разделены, сеть называют *двусторонней* (two-sided). При совмещенных входах и выходах сеть является *односторонней* (one-sided).

Обычно ключи в динамических СМС группируются в так называемые *ступени коммутации*. В зависимости от того, сколько ступеней коммутации содержит сеть, она может быть *одноступенчатой* или *многоступенчатой*. Наличие более чем одной ступени коммутации позволяет обеспечить множественность путей между любыми парами входов и выходов.

Блокирующие и неблокирующие многоуровневые сети

Минимальным требованием к сети с коммутацией является поддержка соединения любого входа с любым выходом. Для этого в сети с n входами и n выходами система ключей обязана предоставить n \ вариантов коммутации входов и выходов

(перестановок - permutations). Проблема усложняется, когда сеть должна обеспечивать одновременную передачу данных между многими парами терминальных узлов (multicast), причем так, чтобы не возникали конфликты (блокировки) из-за передачи данных через одни и те же коммутирующие элементы в одно и то же время. Подобные топологии должны поддерживать n" перестановок. С этих позиций все топологии СМС с коммутацией разделяются на три типа: неблокирующие, неблокирующие с реконфигурацией и блокирующие.

В *неблокирующих сетях* обеспечивается соединение между любыми парами входных и выходных терминалов без перенастройки коммутирующих элементов сети, В рамках этой группы различают сети строго неблокирующие (strictly non-blocking) и неблокирующие в широком смысле (wide sense non-blocking), В *строго неблокирующих сетях* возникновение блокировок принципиально невозможно в силу примененной топологии. К таким относятся матричная сеть и сеть Клоша. *Неблокирующими в широком смысле* называют топологии, в которых конфликты при любых соединениях не возникают только при соблюдении определенного алгоритма маршрутизации.

В *неблокирующих сетях с реконфигурацией* также возможна реализация соединения между произвольными входными и выходными терминалами, но для этого необходимо изменить настройку коммутаторов сети и маршрут связи между соединенными терминалами. Примерами таких сетей служат сети Бенеша, Бэтчера, «Мемфис» и др.

В *блокирующих сетях,* если какое-либо соединение уже установлено, это может стать причиной невозможности установления других соединений. К блокирующим относятся сети «Баньяна, «Омега», n-куб и др.

Шинная топология

Сети с шинной архитектурой - наиболее простой и дешевый вид динамических сетей. *При одношинной топологии,* показанной на рис. 12,17, а, все узлы имеют порядок 1 (d = 1) и подключены к одной совместно используемой шине. В каждый момент времени обмен сообщениями может вести только одна пара узлов, то есть на период передачи сообщения шину можно рассматривать как сеть, состоящую из двух узлов, в силу чего ее диаметр всегда равен 1 (D = 1), Также единице равна и ширина бисекции *(B),* поскольку топология допускает одновременную передачу только одного сообщения. Однотипная конфигурация может быть полезной, когда число узлов невелико, то есть когда трафик шины мал по сравнению с ее пропускной способностью. Одношинную архитектуру часто используют для объединения нескольких узлов в группу (кластер), после чего из таких кластеров образуют сеть на базе других видов топологии.

Многошинная топология предполагает наличие *n* независимых шин и подключение узлов к каждой из этих шин (рис, 12.17, *б),* что позволяет вести одновременную пересылку сообщений между *n* парами узлов. Такая топология вполне пригодна для высокопроизводительных ВС. Диаметр сети по-прежнему равен 1, в то время как пропускная способность возрастает пропорционально числу шин. По сравнению с одношинной архитектурой управление сетью с несколькими шинами сложнее из-за необходимости предотвращения конфликтов, возникающих, когда

в парах узлов, обменивающихся по разным шинам, присутствует общий узел. Кроме того, с увеличением порядка узлов сложнее становится их техническая реализация.

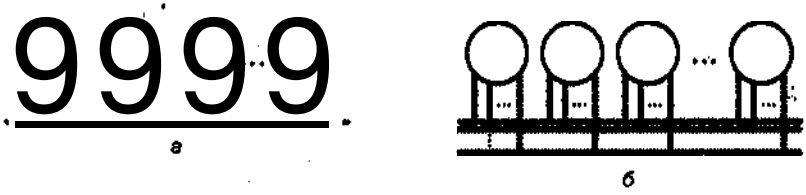


Рис. 12.17. Шинная топология: а -с одной шиной; б - с о многими шинами

Топология перекрестной коммутации («кроссбар»)

Топология перекрестной коммутации мультипроцессорной системы (crossbar switch system) на основе матричного (координатного) коммутатора представляет собой классический пример одноступенчатой динамической сети. Не совсем официальный термин «кроссбар», который будет применяться в дальнейшем для обозначения данной топологии, берет свое начало с механических координатных (шаговых) искателей, использовавшихся на заре, телефонии. Кроссбар nxm (рис. 12.18) представляет собой коммутатор, способный соединить *n* входных и *m* выходных терминальных узлов с уровнем параллелизма, равным min (n, m). Главное достоинство рассматриваемой топологии состоит том, что сеть получается неблокирующей и обеспечивает меньшую задержку в передаче сообщений по сравнению с другими топологиями, поскольку любой путь содержит только один ключ. Тем не менее, из-за того, что число ключей в сети равно *n x m,* использование кроссбара в больших сетях становится непрактичным, хотя это достаточно хороший выбор для малых сетей. Ниже будет показано, что для больших неблокирующих сетей можно предложить иные топологии, требующие существенно меньшего количества ключей.

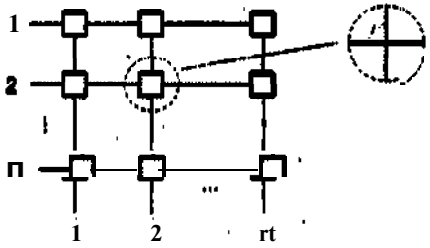


Рис. 12.18. Матричный коммутатор n x m

Когда $n = m$, о такой ситуации говорят *«полный кроссбар»* (присвоим мужской род, хотя, как и все неодушевленное, в английском это средний род). «Полный кроссбар»- на n входов и *n* выходов содержит n^2 ключей. Диаметр сети равен 1, ширина бисекции - n/2. Этот вариант часто используют в сетях с древовидной топологией

для объединения узлов нижнего уровня, роль которых играют небольшие группы (кластеры) процессоров и модулей памяти.

Современные коммерчески доступные матричные коммутаторы способны соединять до 256 устройств. Топология используется для организации соединений в некоторых серийно выпускаемых вычислительных системах, например в Fujitsu VPP500 224x224.

Коммутирующие элементы сетей с динамической топологией

Поскольку последующие рассматриваемые топологии относятся к многоступенчатым, сначала необходимо определить типы коммутирующих элементов, применяемых в ступенях коммутации таких сетей. По этому признаку различают:

- сети на основе перекрестной коммутации;
- сети на основе базового коммутирующего элемента.

В сетях, относящихся к первой группе, в качестве базового коммутирующего элемента используется кроссбар *n* x *m*. Для второй категории роль коммутирующего элемента играет -«полный кроссбар» 2x2. Потенциально такой коммутатор управляется четырехразрядным двоичным кодом и обеспечивает 16 вариантов коммутации, из которых полезными можно считать 12. На практике же обычно задействуют только четыре возможных состояния кроссбара 2x2, которые определяются двухразрядным управляющим кодом (рис. 12.19). Подобный кроссбар называют *базовым коммутирующим элементом* (БКЭ) или *b-элементом*. Первые два состояния БКЭ являются основными: в них входная информация может транслироваться на выходы прямо либо перекрестно. Два следующих состояния предназначены для широковещательного режима, когда сообщение от одного узла одновременно транслируется на все подключенные к нему прочие узлы. Широковещательный режим используется редко. Сигналы на переключение БКЭ в определенное состояние могут формироваться устройством управления сетью. В более сложном варианте БКЭ эти сигналы формируются внутри самого *b*-элемента, исходя из адресов пунктов назначения, содержащихся во входных сообщениях.

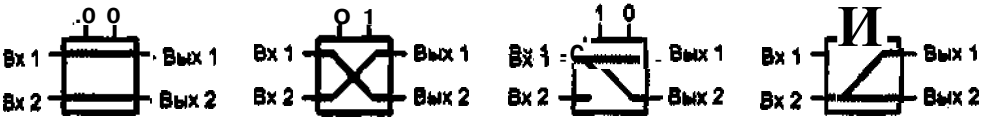


Рис. 12.19. Состояния *b*-элемента

Структура *b*-элемента показана на рис. 12.20.

Выбор в пользу того или иного варианта коммутации входных сообщений (пакетов) осуществляется схемой логики принятия решения *b*-элемента. Конкретный вид коммутации реализуется сдвоенным мультиплексором, управляемым с выхода защелки, где хранится результат работы схемы логики принятия решения. Элементы задержки обеспечивают синхронизацию процессов принятия решения и пересылки пакетов с входов на выходы.

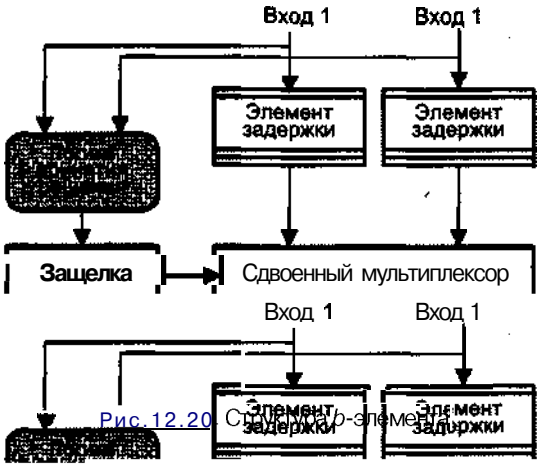


Рис. 12.20. Структура *b*-элемента

Сложность *b*-элемента находится в зависимости от логики принятия решения. В ряде архитектур БКЭ их состояние определяется только битом активности пакета. В иных архитектурах используются адреса источника и получателя данных, хранящиеся в заголовке пакета , что может потребовать поддержания в БКЭ специальных таблиц. Тем не менее во всех своих вариантах *b*-элементы достаточно просты, что позволяет реализовать их на базе интегральных микросхем.

Топология «Баньян»

Данный вид сети получил свое название из-за того, что его схема напоминает воздушные корни дерева баньян (индийской смоковницы) [98]. В топологии «Баньян» между каждой входной и выходной линиями существует только один путь [156]. Сеть $n \times n \ (n - 2^m)$ состоит из $mn/2$ базовых коммутирующих элементов.

Сеть «Баньян» 4x4 по топологии совпадаете сетью «Баттерфляй». На рис. 1221 показана сеть «Баньян» 8x8. Передаваемый пакет в своем заголовке содержит трехразрядный двоичный номер узла назначения. Данная сеть относится к *сетям* с *самомаршрутизацией* (self-routing), поскольку адрес пункта назначения не только определяет маршрут сообщения к нужному узлу, но и используется для управления прохождением сообщения по этому маршруту. Каждый БКЭ, куда попадает пакет, просматривает один бит адреса и в зависимости от его значения направляет сообщение на выход 1 или 2. Состояние *b*-элементов первой ступени сети (левый столбец БКЭ) определяется старшим битом адреса узла назначения. Средней ступенью (второй столбец) управляет средний бит адреса, а третьей ступенью (правый столбец) - младший бит. Если значение бита равно 0, то сообщение пропускается через верхний выход БКЭ, а при единичном значении — через нижний. На рисунке показан маршрут сообщения с входного узла 2 (0102) к выходному узлу 5 (1012). Адрес узла назначения содержится в заголовке сообщения.

Топология «Баньян» весьма популярна из-за того, что коммутация обеспечивается простыми БКЭ, работающими с одинаковой скоростью, сообщения передаются параллельно. Кроме того, большие сети могут быть построены из стандартных модулей меньшего размера.

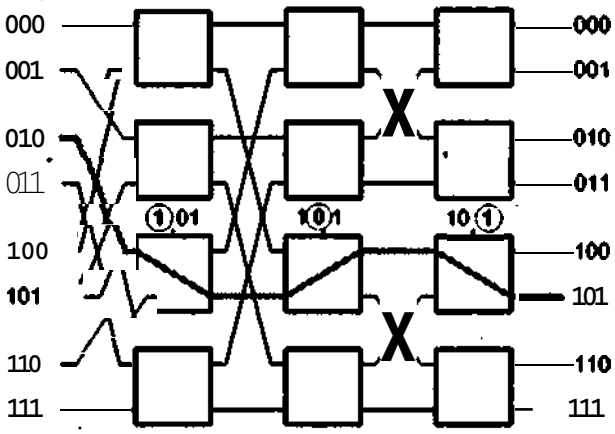


Рис. 12.21. Топология типа «Баньян»

Топология «Омега»

Сеть с топологией «Омега» по сути является подклассом «баньян»-сетей [71,148, 203] и представляет собой многоуровневую структуру, где смежные уровни связаны между собой согласно функции идеального тасования. Сеть nxn, где $n = 2^m$, состоит из *m* уровней БКЭ, при общем числе БКЭ - *mn/2.* Количество соединений, обеспечиваемых сетью «Омега», равно $n^{n/2}$, что гораздо меньше, чем n!, то есть топология «Омега» является блокирующей. Так, при $n = 8$ процент комбинаций, возможных в сети «Омега», по отношению к потенциально допустимому числу комбинаций составляет $\frac{8^4}{8!} = \frac{4096}{40320} \approx 0{,}1016$, или 10,16%.

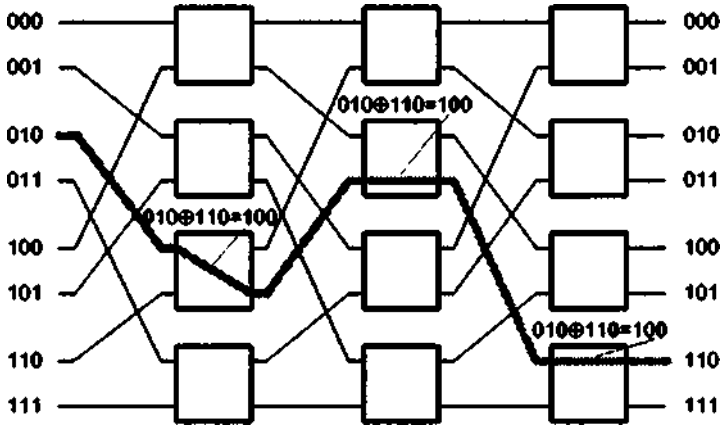


Рис. 12.22. Сеть с топологией «Омега»

Рассмотрим порядок установки *b*-элементов сети для соединения входного и выходного терминальных узлов, двоичное n-разрядное представление номеров которых есть соответственно $(a_n a_{n-1} \ldots a_1)$ И $(b_n b_{n-1} \ldots b_1)$. Состояние, в которое пе-

реключается БКЭ на i-й ступени, определяется с помощью операции сложения по модулю 2 значений i-го бита в адресах входного и выходного терминальных узлов. Если $a_i \oplus b_i = 0$, то БКЭ, расположенный на *i-й* ступени сети, обеспечивает прямую связь входа с выходом, а при $a_p \oplus b_p \bullet 1$ — перекрестное соединение. На рис. 12.22 показан процесс прохождения сообщения по сети «Омега» 8 х 8 от входного терминала 2 (010_2) к выходному терминалу 6(110_2). Таким образом, если в сообщении присутствуют адреса источника и получателя сообщений, сеть может функционировать в режиме самомаршрутизации.

Топология «Дельта»

Важный подкласс «баньян»-сетей образуют сети «Дельта», предложенные Пателом в 1981 году [176]. В них основание системы счисления при адресации узлов для маршрутизации может отличаться от 2. Сеть соединяет a^n входов с b^n выходами посредством *n* ступеней кроссбаров *a x b (в* сетях с такими топологиями, как «Омега», «базовая линиям и «косвенный» n-куб, используется двоичная система счисления, то есть $a = 2$ и $b = 2$). Адрес получателя задается в заголовке сообщения числом в системе счисления с основанием *b,* а для прохождения сообщения по сети организуется самомаршрутизация. Каждая цифра адреса имеет значение *в* диапазоне от 0 до b - 1 и выбирает один из b выходов коммутирующего элемента типа кроссбар *a* х *b.* Пример сети «Дельта» показан на рис. 12.23, *а.*

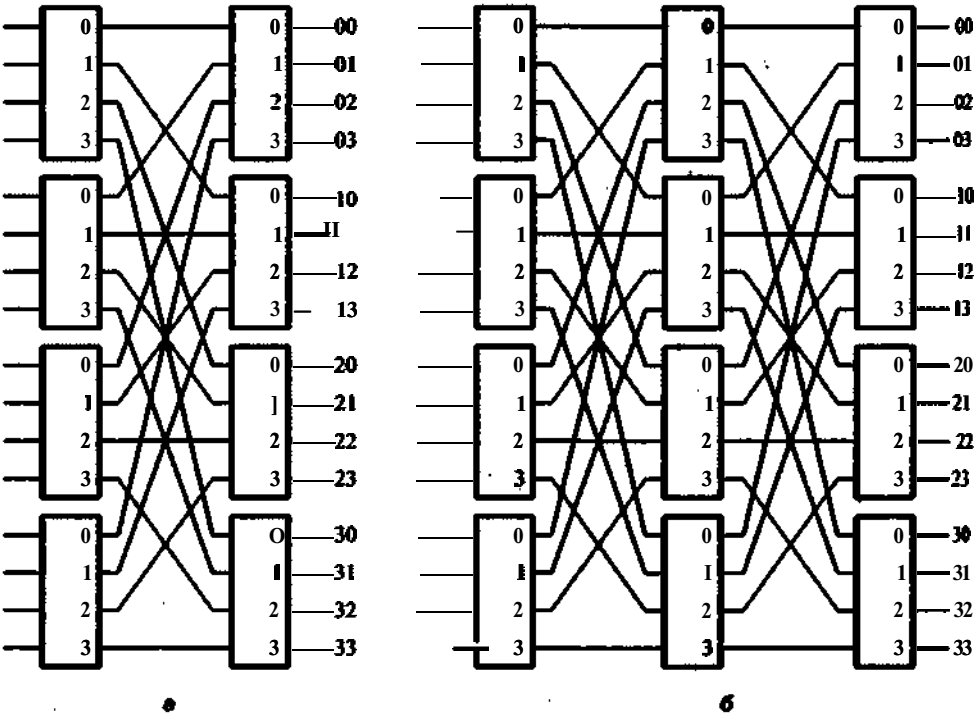


Рис. 1 2.23. Структура сети «Дельта»: *а* - по базе 4; *б* - с дополнительной ступенью

В отличие от сети «Омега» входы не подвергаются тасованию. Это не влияет на алгоритм маршрутизации, поскольку важность имеет не адрес источника, а адрес получателя. На рис. 12.23, *а* связь между ступенями соответствует идеальному тасованию — коммутаторы соединены так, что для связи любого входа с любым выходом образуется единственный путь, причем пути для любой пары равны по длине. В сеть «Дельта» могут быть введены и дополнительные ступени (рис. 12.23, *б),* чтобы обеспечить более чем один маршрут от входа к выходу.

Топология Бенеша

Как уже отмечалось, в топологии «Баньян» между каждой входным и выходным терминалом существует только один путь. С добавлением к такой сети дополнительной ступени БКЭ число возможных маршрутов удваивается. Дополнительные пути позволяют изменять трафик сообщения с целью устранения конфликтов. При добавлении к сети -«Баньян»*(m* - 1)-го уровня, где я - 2^m, получаем топологию Бенеша (рис. 12.24) [152,172]. В сети Бенеша *n* x *n* число ступеней определяется выражением *2m* - 1, а число БКЭ равно - $n/2(2m-1)$.

Сеть Бенеша с *n* входами и n выходами имеет симметричную структуру, в каждой половине которой (верхней и нижней) между входными и выходными БКЭ расположена такая же сеть Бенеша, но с *n/2* входами и n/2 выходами.

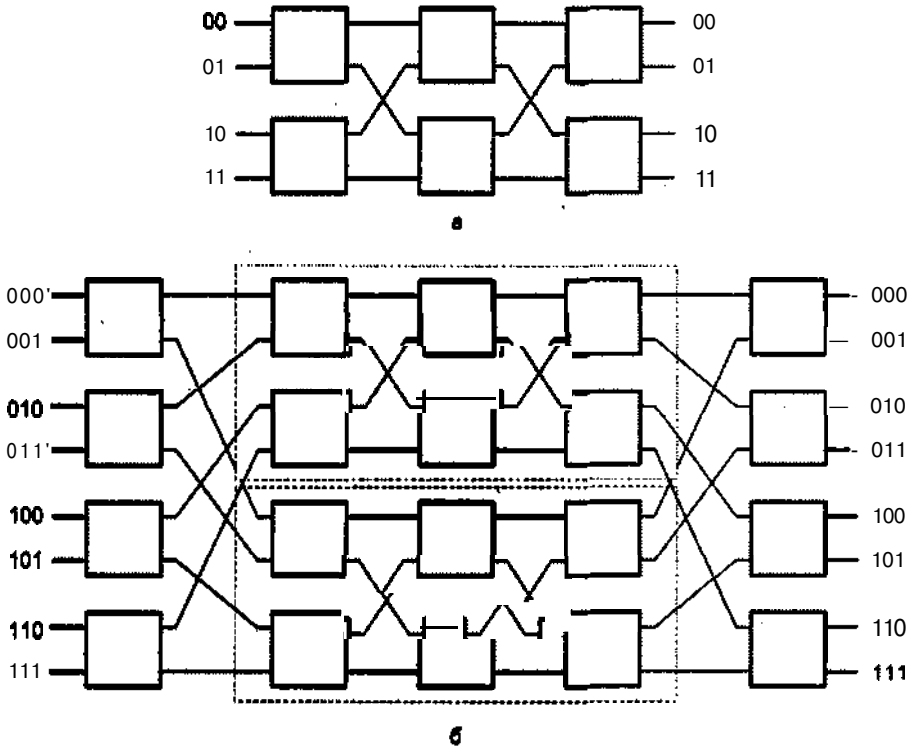


Рис. 12.24. Топология Бенеша: а - *4 х 4; б* - 8x8

Рассматриваемая топология относится к типу неблокирующих сетей с реконфигурацией.

Топология Клоша

В 1953 году Клош показал, что многоступенчатая сеть на основе элементов типа кроссбар, содержащая не менее трех ступеней, может обладать характеристиками неблокирующей сети [62,76/.

Сеть Клоша с тремя ступенями, показанная на рис. 12.25, содержит r_1, кроссбаров во входной ступени, *m* кроссбаров в промежуточной ступени и r_2 кроссбаров в выходной ступени. У каждого коммутатора входной ступени есть n_1 входов и *m* выходов - по одному выходу на каждый кроссбар промежуточной ступени. Коммутаторы промежуточной ступени имеют r_1, входов по числу кроссбаров входной ступени и r_2 выходов, что соответствует количеству переключателей в выходной ступени сети. Выходная ступень сети строится из кроссбаров с *m* входами и n_2 выходами. Отсюда числа n_2 ,n_2, r_1 ,r_2 *и m* полностью определяют сеть. Число входов сети $N = r_1 n_1$ а выходов — $M = r_2 n_2$.

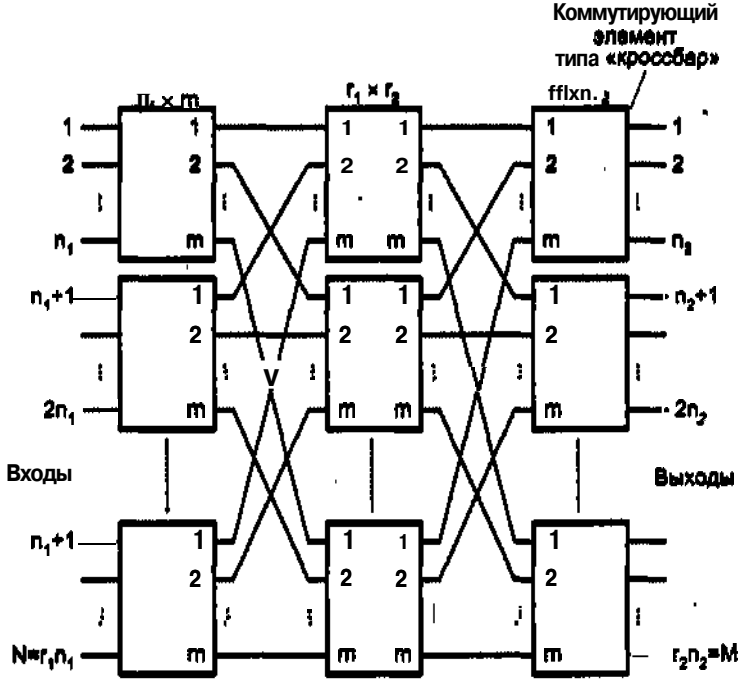


Рис. 12.25. Трехступенчатая сеть с топологией Клоша

Связи внутри составного коммутатора организованы по следующим правилам:

- *к-й* выход i-го входного коммутатора соединен с i-м входом k-го промежуточного коммутатора;
- *k-й* вход j-го выходного коммутатора соединен *с j-м* выходом k-го промежуточного коммутатора.

Каждый модуль первой и третьей ступеней сети соединен с каждым модулем второй ее ступени.

Хотя в рассматриваемой топологии обеспечивается путь от любого входа к любому выходу, ответ на вопрос, будет ли сеть неблокирующей, зависит от числа промежуточных.звеньев. Клош доказал, что подобная сеть является неблокирующей, если количество кроссбаров в промежуточной ступени m удовлетворяет условию: $m = n_2 + n_2 - 1$. Если n1 = n_2, то матричные переключатели в промежуточной ступени представляют собой «полный кроссбар» и критерий неблокируемости приобретает вид: m — 2n-\. При условии $m = n_2$ *сеть* Клоша можно отнести к неблокирующим сетям с реконфигурацией. Во всех остальных случаях данная топология становится блокирующей.

Вычислительные системы, в которых соединения реализованы согласно топологии Клоша, выпускают многие фирмы, в частности Fujitsu (FETEX-150), Nippon Electric Company (ATOM), Hitachi. Частный случай сети Клоша при $n_1 = r_1$,=$r_2 = n_2$ называется сетью «Мемфис». Топология «Мемфис» нашла применение в вычислительной системе GF-11 фирмы IBM.

Топология двоичной n-кубической сети с косвенными связями

На рис. 12.26 показана косвенная двоичная n-кубическая сеть 8 x 8 [180,188].

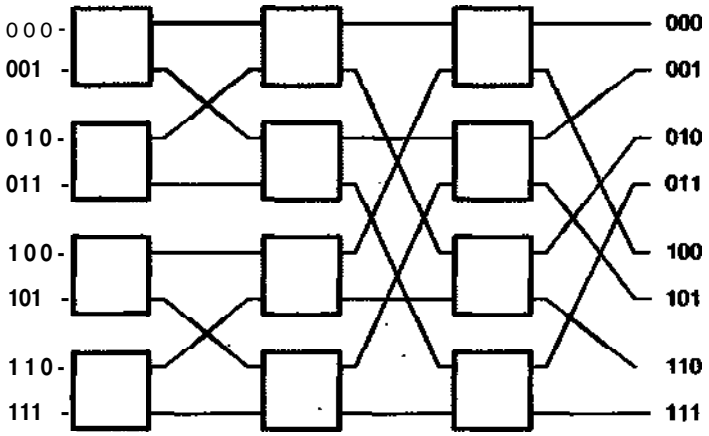


Рис. 12.26. Топология двоичной n-кубической сети

Здесь ступени коммутации связаны по топологии «Баттерфляй», а на последней ступени используется функция идеального тасования. Фактически сеть представляет собой обращенную матрицу сети «Омега», В этом можно убедиться, если соответствующим образом поменять местами БКЭ в каждом уровне сети «Омега», за исключением первого и последнего.

Топология базовой линии

Данный вид сети представляет собой многоступенчатую топологию, где в качестве коммутаторов служат *b*-элементы (12.27). Топология обеспечивает очень удоб-

ный алгоритм самомаршрутизации, в котором последовательные ступени коммутаторов управляются последовательными битами адреса получателя. Каждая ступень сети на принципе базовой линии делит возможный диапазон маршрутов пополам. Старший бит адреса назначения управляет первой ступенью. При нулевом значении этого бита сообщения с любого из входов поступят на вторую ступень сети с верхних выходов БКЭ первой ступени, то есть они смогут прийти только на верхнюю половину выходов (в нашем примере это выходы с номерами 000-011), а при единичном значении бита— на нижнюю половину выходов (100-111). Второй бит адреса назначения управляет коммутаторами второй ступени, которая делит половину выходов, выбранную первой ступенью, также пополам. Процесс повторяется на последующих ступенях до тех пор, пока младший бит адреса назначения на последней ступени не выберет нужный выход сети. Таким образом, сеть на 8 входов требует наличия трех ступеней коммутации, сеть на 16 входов — 4 ступеней и т. д.

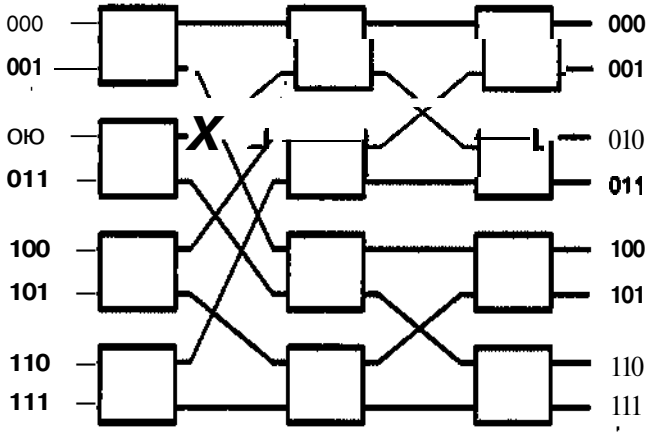


Рис. 12.27.Топология базовой линии

Как видно из рисунка, сеть с топологией базовой линии совпадает с первыми *m (n* - 2^m) уровнями сети Бенеша на *n* входов и *n* выходов. Если к последнему уровню этой сети добавить сеть с инверсной перестановкой битов, то получим так называемую *R-сеть.* Сеть с инверсной перестановкой битов имеет фиксированные связи входного терминала ($a_m a_{m1}... a_1$) с выходным терминалом ($a_1, a_2... a_m$)и фактически представляет собой косвенную двоичную n-кубическую сеть.

Контрольные вопросы

1. Прокомментируйте классификацию сетей по топологии, функциональности узлов, изменяемости взаимосвязей, синхронизации, коммутации и управлению.
2. Дайте развернутую характеристику метрик, описывающих соединения сети.
3. Сформулируйте достоинства и недостатки наиболее известных функций маршрутизации данных.
4. Обоснуйте достоинства и недостатки линейной топологии сети.

5. Дайте характеристику плюсов и минусов кольцевой топологии сети. Какие варианты этой топологии практически используются?
6. Проведите сравнительный анализ звездообразной и древовидной топологий сети.
7. Выполните сравнительный анализ известных вариантов решетчатой топологии сети.
8. Поясните смысл топологии k-ичного n-куба сети. Как строится такой куб?
9. Дайте сравнительную характеристику блокирующих и неблокирующих многоуровневых сетей.
10. Проведите сравнительный анализ одношинной и многошинной топологий динамических сетей, акцентируя их сильные и слабые стороны.
11. Сравните популярные разновидности «баньян»-сетей: «Омега», «Дельта»-. Можно ли причислить к этому классу сети Бенеша, и если можно, то почему?
12. Дайте развернутое объяснение отличий топологии Клоша от «баньян»-сетей. Можно ли найти у них сходные черты, и если можно, то какие?
13. С какой топологией сходна организация двоичной n-кубической сети с косвенными связями? Ответ обоснуйте, приведя конкретный пример.
14. Охарактеризуйте смысл топологии на принципе базовой линии. Изобразите структуру соответствующей сети на 20 входов.

Глава 13

Вычислительные системы класса SIMD

SIMD-системы были первыми вычислительными системами, состоящими из большого числа процессоров, и среди первых систем, где была достигнута производительность порядка GFLOPS. Согласно классификации Флинна, к классу SIMD относятся ВС, где множество элементов данных подвергается параллельной, но однотипной обработке. SIMD-системы во многом похожи на классические фоннеймановские ВМ: в них также имеется одно устройство управления, обеспечивающее последовательное выполнение команд программы. Различие касается стадии выполнения, когда общая команда транслируется множеству процессоров (в простейшем случае - АЛУ), каждый из которых обрабатывает свои данные. На рис. 13.1 показаны приблизительные характеристики производительности некоторых типов вычислительных систем класса SIMD.

Ранее уже отмечалась нечеткость классификации Флинна, из-за чего разные типы ВС могут быть отнесены к тому или иному классу. Тем не менее в настоящее

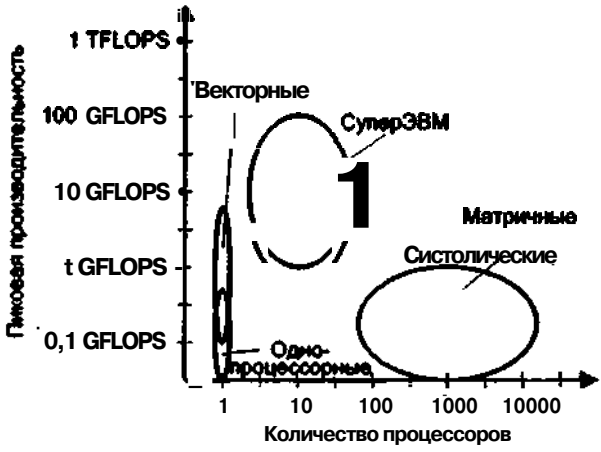


Рис. 1 3 . 1 . Производительность SIMO-систем как функция их типа и количества процессоров

время принято считать, что класс SIMD составляют векторные (векторно-конвейерные), матричные, ассоциативные, систолические и VLIW-вычислительные системы. Именно эти ВС и будут предметом рассмотрения в настоящем разделе.

Векторные и векторно-конвейерные вычислительные системы

Хотя производительность ВС общего назначения неуклонно возрастает, по-прежнему остаются задачи, требующие существенно большей вычислительной мощности. К таким, прежде всего, следует отнести задачи моделирования реальных процессов и объектов, для которых характерна обработка больших регулярных массивов чисел в форме с плавающей запятой. Такие массивы представляются матрицами и векторами, а алгоритмы их обработки описываются в терминах матричных операций. Как известно, основные матричные операции сводятся к однотипным действиям над парами элементов исходных матриц, которые, чаще всего, можно производить параллельно, В универсальных ВС, ориентированных на скалярные операции, обработка матриц выполняется поэлементно и последовательно. При большой размерности массивов последовательная обработка элементов матриц занимает слишком много времени, что и приводит к неэффективности универсальных ВС для рассматриваемого класса задач. Для обработки массивов требуются вычислительные средства, позволяющие с помощью единой команды производить действие сразу над всеми элементами массивов - средства *векторной обработки*.

Понятие вектора и размещение данных в памяти

В средствах векторной обработки под *вектором* понимается одномерный массив однотипных данных (обычно в форме с плавающей запятой), регулярным образом размещенных в памяти ВС. Если обработке подвергаются многомерные массивы, их также рассматривают как векторы. Такой подход допустим, если учесть, каким образом многомерные массивы хранятся в памяти ВМ Пусть имеется массив данных Л, представляющий собой прямоугольную матрицу размерности 4x5 (рис. 13.2).

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a»,	a_{32}	B»	a_{34}	a_{35}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}

Рис. 13.2. Прямоугольная матрица данных

При размещении матрицы в памяти все ее элементы заносятся в ячейки с последовательными адресами, причем данные могут быть записаны строка за строкой или столбец за столбцом (рис. 13.3). С учетом такого размещения многомерных массивов в памяти вполне допустимо рассматривать их как векторы и ориентировать соответствующие вычислительные средства на обработку одномерных массивов данных (векторов).

Рис. 13.3. Способы размещения в памяти матрицы 4x5

Действия над многомерными массивами имеют свою специфику. Например, в двумерном массиве обработка может вестись как по строкам, так и по столбцам. Это выражается в том, с каким шагом должен меняться адрес очередного выбираемого из памяти элемента. Так, если рассмотренная в примере матрица расположена в памяти построчно, адреса последовательных элементов строки различаются на единицу, а для элементов столбца шаг равен пяти. При размещении матрицы по столбцам единице будет равен шаг по столбцу, а шаг по строке - четырем. В векторной концепции для обозначения шага, с которым элементы вектора извлекаются из памяти, применяют термин *шаг по индексу* (stride).

Еще одной характеристикой вектора является число составляющих его элементов - *длина вектора*.

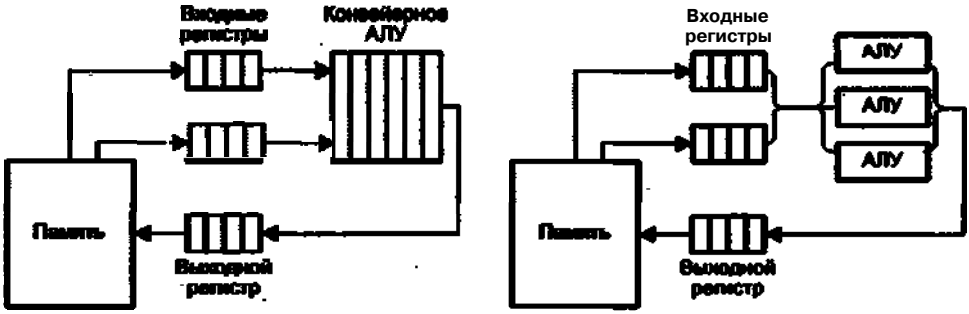
Понятие векторного процессора

Векторный процессор — это процессор, в котором операндами некоторых команд могут выступать упорядоченные массивы данных — векторы. Векторный процессор может быть реализован в двух вариантах. В первом он представляет собой дополнительный блок к универсальной вычислительной машине (системе). Во втором - векторный процессор - это основа самостоятельной ВС.

Рассмотрим возможные подходы к архитектуре средств векторной обработки. Наиболее распространенные из них сводятся к трем группам:

* конвейерное АЛУ;
* массив АЛУ;
* массив процессорных элементов.

Последний вариант - один из случаев многопроцессорной системы, известной *как матричная ВС*. Понятие векторного процессора имеет отношение к двум первым группам, причем, как правило, к первой. Эти две категории иллюстрирует рис. 13.4 [200].



в б

Рис. 13.4. Варианты векторных вычислений: *а* - с конвейерным АЛУ; *б -с* несколькими АЛУ

В варианте с конвейерным АЛУ (рис. 13.4, а) обработка элементов векторов производится конвейерным АЛУ для чисел с плавающей запятой (ПЗ). Операции с числами в форме с ПЗ достаточно сложны, но поддаются разбиению на отдельные шаги. Так, сложение двух чисел может быть сведено к четырем этапам [200]:

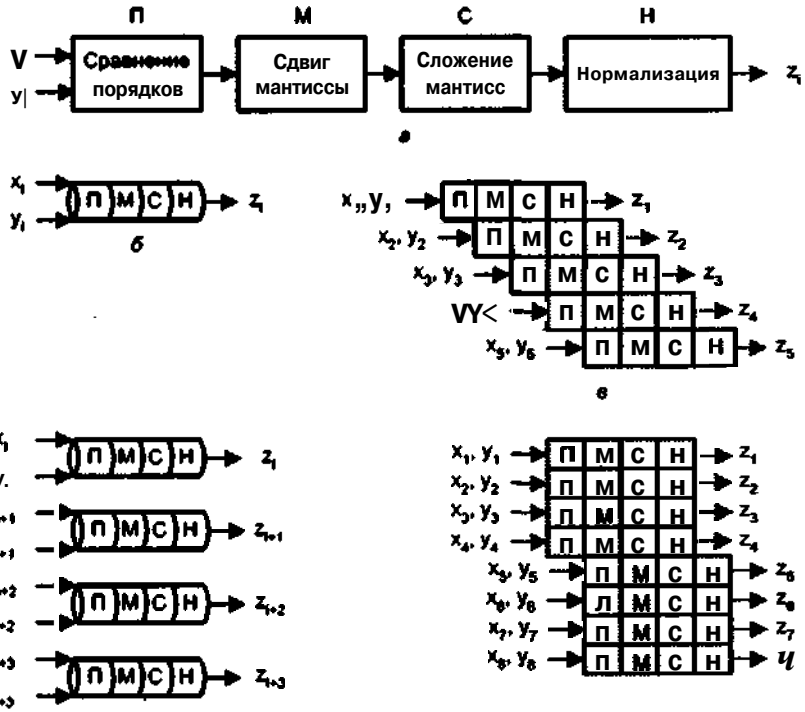


Рис. 13.5. Обработка векторов: *а* - структура арифметического конвейера для чисел с плавающей запятой; б - обозначение конвейера; *в* - обработка векторов в конвейерном АЛУ; г- параллельная обработка векторов несколькими конвейерными АЛУ; *д* - конвейерная обработка векторов четырьмя АЛУ

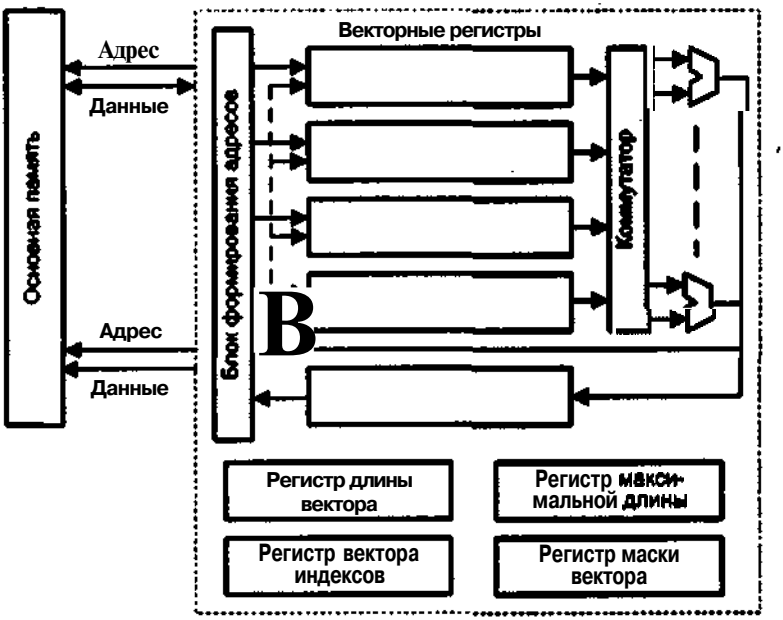
сравнению порядков, сдвигу мантиссы меньшего из чисел, сложению мантисс и нормализации результата (рис. 13.5, *а*). Каждый этап может быть реализован с помощью отдельной ступени конвейерного АЛУ (рис. 13.5,б). Очередной элемент вектора подается на вход конвейера, как только освобождается первая ступень (рис. 13.5, в). Ясно, что такой вариант вполне годится для обработки векторов.

Одновременные операции над элементами векторов можно проводить и с помощью нескольких параллельно используемых АЛУ, каждое из которых отвечает за одну пару элементов (см. рис. 13.4,*б*). Такого рода обработка, когда каждое из АЛУ является конвейерным, показана на рис. 13.5, *г*.

Если параллельно используются конвейерные АЛУ, то возможен еще один уровень конвейеризации, что иллюстрирует рис. 13.5, *д*. Вычислительные системы, где реализована эта идея, называют *векторно-конвейерными*. Коммерческие векторно-конвейерные ВС, в состав которых для обеспечения универсальности включен также скалярный процессор, известны как *суперЭВМ*.

Структура векторного процессора

Обобщенная структура векторного процессора приведена на рис. 13.6. На схеме показаны основные узлы процессора, без детализации некоторых связей между ними.



Рис, 13.6. Упрощенная структура векторного процессора

Обработка всех *n* компонентов векторов-операндов задается одной *векторной команд*ой. Общепринято, что элементы векторов представляются числами в форме с плавающей запятой (ПЗ). АЛУ векторного процессора может быть реализовано в виде единого конвейерного устройства, способного выполнять все предусмотренные операции над числами с ПЗ. Более распространена, однако, иная

структура, — в ней АЛУ состоит из отдельных блоков сложения и умножения, а иногда и блока для вычисления обратной величины, когда операция деления

- реализуется в виде $X(1/Y)$. Каждый из таких блоков также конвейеризирован.
Y

Кроме того, в состав векторной вычислительной системы обычно включают и скалярный процессор, что позволяет параллельно выполнять векторные и скалярные команды.

Для хранения векторов-операндов вместо множества скалярных регистров используют так называемые *векторные регистры,* которые представляют собой совокупность скалярных регистров, объединенных в очередь типа FIFO, способную хранить 50-100 чисел с плавающей запятой. Набор векторных регистров (V_a, V_b, V_c,...) имеется в любом векторным процессоре. Система команд векторного процессора поддерживает работу с векторными регистрами и обязательно включает в себя команды:

* загрузки векторного регистра содержимым последовательных ячеек памяти, указанных адресом первой ячейки этой последовательности;
- выполнения операций над всеми элементами векторов, находящихся в векторных регистрах;
- сохранения содержимого векторного регистра в последовательности ячеек памяти, указанных адресом первой ячейки этой последовательности.

Примером одной из наиболее распространенных операций, возлагаемых на векторный процессор, может служить операция перемножения матриц [161]. Рассмотрим перемножение двух матриц A и B размерности 3x3.

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}.$$

Элементы матрицы результата C связаны с соответствующими элементами исходных матриц A и B операцией скалярного произведения:

$$c_{ij} = \sum_{k=1}^{3} a_{ik} \times b_{kj}.$$

Так, элемент c_{11} вычисляется как

$$c_{11} = a_{11} \text{ x } b_{11} + a_{12} \times b_{21} + a_{13} \text{x } b_{31}.$$

Это требует трех операций умножения и (после инициализации c_{11} нулем) трех операций сложения. Общее число умножений и сложений для рассматриваемого примера составляет 9 x 3 = 27. Если рассматривать связанные операции умножения и сложения как одну кумулятивную операцию $c + a \times b$, то для умножения двух матриц $n \times n$ необходимо n^3 операций типа «умножение-сложением»Вся процедура сводится к получению n^2 скалярных произведений, каждое из которых является итогом « операций «умножение-сложение», учитывая, что перед вычислением каждого элемента c_{ij} его необходимо обнулить. Таким образом, скалярное произведение состоит из k членов:

$$C - A_1B_1 + A_2B_2 + A_3B_3 + A_4B_4 + ... + A_kB_k.$$

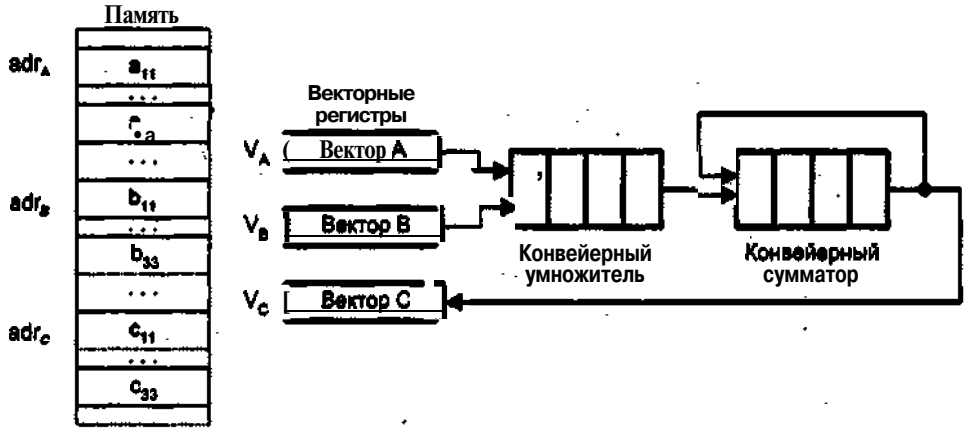


Рис. 13.7. Векторный процессор для вычисления скалярного произведения

Векторный процессор с конвейеризированными блоками обработки для вычисления скалярного произведения показан на рис. 13.7.

Векторы A и B, хранящиеся в памяти начиная с адресов adr_A u adr_B, загружаются в векторные регистры V_A и V_B соответственно. Предполагается, что конвейерные умножитель и сумматор состоят из четырех сегментов, которые вначале инициализируются нулем, поэтому в течение первых восьми циклов, пока оба конвейера не заполнятся, на выходе сумматора будет 0. Пары (A_i, *B)* подаются на вход умножителя и перемножаются в темпе одна пара за цикл. После первых четырех циклов произведения начинают суммироваться с данными, поступающими с выхода сумматора. В течение следующих четырех циклов на вход сумматора поступают суммы произведений из умножителя с нулем. К концу восьмого цикла в сегментах сумматора находятся четыре первых произведения A_1B_1, ..., A_4B_4, а в сегментах умножителя — следующие четыре произведения: A_5B_5,..., A_8B_8. К началу девятого цикла на выходе сумматора будет A_1B_1, а на выходе умножителя — A_5B_5. Таким образом, девятый цикл начнется со сложения в сумматоре A_1B_1 и A_5B_5. Десятый цикл начнется со сложения A_2B_2 + A_6B_6 и т. д. Процесс суммирования в четырех секциях выглядит так:

$$\begin{aligned} C - {} & \mathbf{A_1B_1} + \mathbf{A_5B_5} + \mathbf{A_9B_9} + \mathbf{A_{13}B_{13}} + \dots \\ & + \mathbf{A_2B_2} + \mathbf{A_6B_6} + \mathbf{A_{10}B_{10}} + \mathbf{A_{14}B_{14}} + \dots \\ & + \mathbf{A_3B_3} + \mathbf{A_7B_7} + \mathbf{A_{11}B_{11}} + \mathbf{A_{15}B_{15}} + \dots \\ & + \mathbf{A_4B_4} + \mathbf{A_8B_8} + \mathbf{A_{12}B_{12}} + \mathbf{A_{16}B_{16}} + \dots \end{aligned}$$

Когда больше не остается членов для сложения, система заносит в умножитель четыре нуля. Конвейер сумматора в своих четырех сегментах при этом будет содержать четыре скалярных произведения, соответствующие четырем суммам, приведенным в четырех строках показанного выше уравнения. Далее четыре частичных суммы складываются для получения окончательного результата.

Программа для вычисления скалярного произведения векторов A и B, хранящихся в двух областях памяти с начальными адресами adr_A и adr_B, соответственно может выглядеть так:

```
VJoad V_A, adr_A
V_load V_B, adr_B
V_multiply V_C, V_A, V_B.
```

Первые две векторные команды V_load загружают векторы из памяти в векторные регистры V_A и V_B Векторная команда умножения V_multiply вычисляет произведение для всех пар одноименных элементов векторов и записывает полученный вектор в векторный регистр V_c.

Важным элементом любого векторного *процессора (ВП) является регистр длины вектора*. Этот регистр определяет, сколько элементов фактически содержит обрабатываемый в данный момент вектор, то есть сколько индивидуальных операций с элементами нужно сделать. В некоторых ВП присутствует также *регистр максимальной длины вектора,* определяющий максимальное число элементов вектора, которое может быть одновременно обработано аппаратурой процессора. Этот регистр используется при разделении очень длинных векторов на сегменты, длина которых соответствует максимальному числу элементов, обрабатываемых аппаратурой за один прием.

Достаточно часто приходится выполнять такие операции, в которых должны участвовать не все элементы векторов. Векторный процессор обеспечивает данный режим с помощью регистре *маски вектора*. В этом регистре каждому элементу вектора соответствует один бит. Установка бита в единицу разрешает запись соответствующего элемента вектора результата в выходной векторный регистр, а сброс в ноль — запрещает.

Как уже упоминалось, элементы векторов в памяти расположены регулярно, и при выполнении векторных операций достаточно указать значение шага по индексу. Существуют, однако, случаи, когда необходимо обрабатывать только ненулевые элементы векторов. Для поддержки подобных операций в системе команд ВП предусмотрены операции *упаковки/распаковки* (gather/scatter). Операция упаковки формирует вектор, содержащий только ненулевые элементы исходного вектора, а операция распаковки производит обратное преобразование. Обе этих задачи векторный процессор решает с помощью вектора индексов, для хранения которого используется *регистр вектора индексов,* по структуре аналогичный регистру маски. В векторе индексов каждому элементу исходного вектора соответствует один бит. Нулевое значение бита свидетельствует, что соответствующий элемент исходного вектора равен нулю.

Использование векторных команд окупается благодаря двум качествам. Во-первых, вместо многократной выборки одних и тех же команд достаточно произвести выборку только одной векторной команды, что позволяет сократить издержки за счет устройства управления и уменьшить требования к пропускной способности памяти. Во-вторых, векторная команда обеспечивает процессор упорядоченными данными. Когда инициируется векторная команда, ВС знает, что ей нужно извлечь *n* пар операндов, расположенных в памяти регулярным образом. Таким образом, процессор может указать памяти на необходимость начать извлечение таких пар. Если используется память с чередованием адресов, эти пары могут быть получены со скоростью одной пары за цикл процессора и направлены для обработки в конвейеризированный функциональный блок. При отсутствии чередова-

ния адресов или других средств извлечения операндов с высокой скоростью преимущества обработки векторов существенно снижаются.

Структуры типа «память-память» и «регистр-регистр»

Принципиальное различие архитектур векторных процессоров проявляется в том, каким образом осуществляется доступ к операндам. При организации *«память-память»** элементы векторов поочередно извлекаются из памяти и сразу же направляются в функциональный блок. По мере обработки получающиеся элементы вектора результата сразу же заносятся в память. В архитектуре типа *«регистр-регистр»* операнды сначала загружаются в *векторные регистры,* каждый из которых может хранить сегмент вектора, например 64 элемента. Векторная операция реализуется путем извлечения операндов из векторных регистров и занесения результата в векторные регистры.

Преимущество ВП с режимом -«память-память» состоит в возможности обработки длинных векторов, в то время как в процессорах типа -«регистр-регистр» приходится разбивать длинные векторы на сегменты фиксированной длины. К сожалению, за гибкость режима «память-память» приходится расплачиваться относительно большими издержками, известными как *время запуска,* представляющее собой временной интервал между инициализацией команды и моментом, когда первый результат появится на выходе конвейера. Большое время запуска в процессорах типа -«память-память» обусловлено скоростью доступа к памяти, которая намного больше скорости доступа к внутреннему регистру. Однако когда конвейер заполнен, результат формируется в каждом цикле. Модель времени работы векторного процессора имеет вид:

$$T\text{-}s + \alpha \times N,$$

где s — время запуска, α — константа, зависящая от команды (обычно 1/2,1 или 2) и N— длина вектора.

Архитектура типа «память-память» реализована в векторно-конвейерных вычислительных системах Advanced Scientific Computer фирмы Texas Instruments Inc., семействе вычислительных систем фирмы Control Data Corporation, прежде все го, Star 100, серии Cyber 200 и ВС типа ETA-10. Все эти вычислительные системы появились в середине 70-х прошлого века после длительного цикла разработки, но к середине 80-х годов от них отказались. Причиной послужило слишком большое время запуска - порядка 100 циклов процессора. Это означает, что операции с короткими векторами выполняются очень неэффективно, и даже при длине векторов в 100 элементов процессор достигал только половины потенциальной производительности.

В вычислительных системах типа «регистр-регистр» векторы имеют сравнительно небольшую длину (в ВС семейства Cray - 64), но время запуска значительно меньше чем в случае «память-память». Этот тип векторных систем гораздо более эффективен при обработке коротких векторов, но при операциях над длинными векторами векторные регистры должны загружаться сегментами несколько раз. В настоящее время ВП типа «регистр-регистр» доминируют на компьютерном рынке. Это вычислительные системы фирмы Cray Research Inc., в частности

модели Y-MP и C-90. Аналогичный подход заложен в системы фирм Fujitsu, Hitachi и NEC. Время цикла в современных ВП варьируется от 2,5 нс (NEC SX-3) до 4,2 нс (Cray C90), а производительность, измеренная по тесту UNPACK, лежит в диапазоне от 1000 до 2000 MFLOPS (от 1 до 2 GFLOPS).

Обработка длинных векторов и матриц

Аппаратура векторных процессоров типа «регистр-регистр» ориентирована на обработку векторов, длина которых совпадает с длиной векторных регистров (ВР), поэтому обработка коротких векторов не вызывает проблем — достаточно записать фактическую длину вектора в регистр длины вектора.

Если размер векторов превышает емкость ВР, используется техника разбиения исходного вектора на сегменты одинаковой длины, совпадающей с емкостью векторных регистров (последний сегмент может быть короче), и последовательной обработки полученных сегментов. В английском языке этот прием называют *strip-mining*. Процесс разбиения обычно происходит на стадии компиляции, но в ряде ВП данная процедура производится по ходу вычислений с помощью аппаратных средств на основе информации, хранящейся в регистре максимальной длины вектора.

Ускорение вычислений

Для повышения скорости обработки векторов все функциональные блоки векторных процессоров строятся по конвейерной схеме, причем так, чтобы каждая ступень любого из конвейеров справлялась со своей операцией за один такт (число ступеней в разных функциональных блоках может быть различным). В некоторых векторных ВС, например Cray C90, этот подход несколько усовершенствован — конвейеры во всех функциональных блоках продублированы (рис. 13.8).

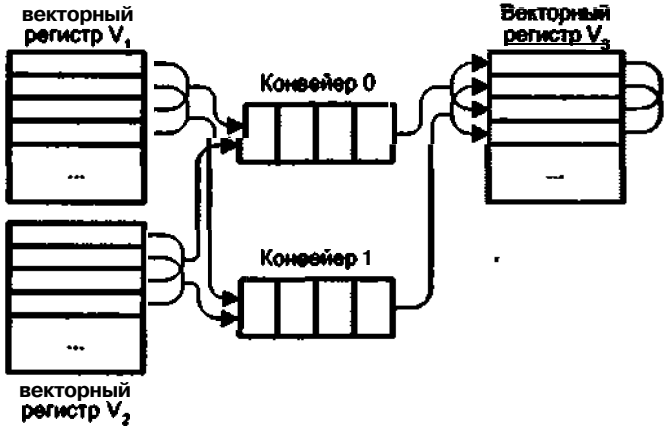


Рис. 13.8. Выполнение векторных операций при двух конвейерах

На конвейер 0 всегда подаются элементы векторов с четными номерами, а на конвейер 1 - е нечетными. В начальный момент на первую ступень конвейера 0 из ВР V1 и V_2 поступают нулевые элементы векторов. Одновременно первые элементы

векторов из этих регистров подаются на первую ступень конвейера 1. На следующем такте на конвейер 0 подаются вторые элементы из V1 и V_2 *а* на конвейер 2 - третьи элементы и т. д. Аналогично происходит распределение результатов в выходном векторном регистре V_3. В итоге функциональный блок при максимальной загрузке в каждом такте выдает не один результат, а два. Добавим, что в скалярных операциях работает только конвейер 0.

Интересной особенностью некоторых ВП типа «регистр-регистр», например ВС фирмы Cray Research Inc., является так называемое *сцепление векторов* (vector chaining или vector linking), когда ВР результата одной векторной операции используется в качестве входного регистра для последующей векторной операции. Для примера рассмотрим последовательность из двух векторных команд, предполагая, что длина векторов равна 64: $V_2 = V_0 \times V_1$, $V_4 = V_2 + V_3$.

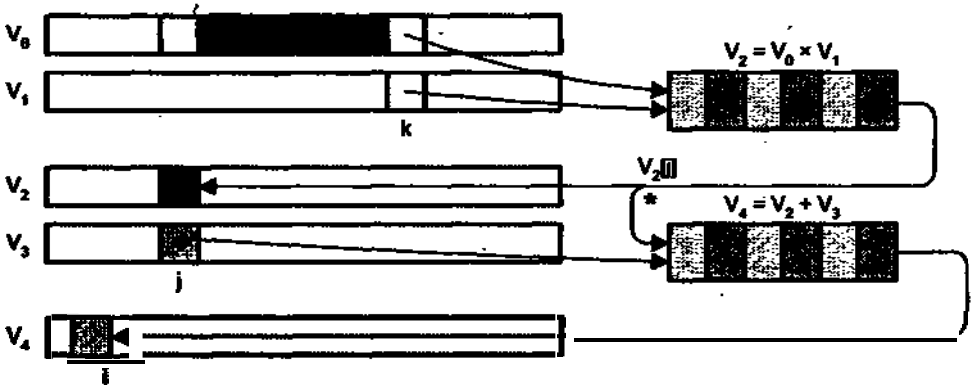


Рис. 13,9. Сцепление векторов

Результат первой команды служит операндом для второй. Напомним, что поскольку команды являются векторными, первая из них должна послать в конвейерный умножитель до 64 пар чисел. Примерно в середине выполнения команды складывается ситуация, когда несколько начальных элементов вектора V_2 будут уже содержать недавно вычисленные произведения; часть элементов V_2 все еще будет находиться в конвейере, а оставшиеся элементы операндов V_0 и V1 еще остаются во входных векторных регистрах, ожидая загрузки в конвейер. Такая ситуация показана на рис. 13.9, где элементы векторов V_0 и V_1, находящиеся в конвейерном умножителе,имеют темную закраску. В этот момент система извлекает элементы $V_0[k]$ и $V_1[k]$ с тем, чтобы направить их на первую ступень конвейера, в то время как $V_2[j]$ покидает конвейер. Сцепление векторов иллюстрирует линия, обозначенная звездочкой. Одновременно с занесением $V_2[j]$ в ВР этот элемент направляется и в конвейерный сумматор, куда также подается и элемент $V_3[j]$. Как видно из рисунка, выполнение второй команды может начаться до завершения первой, и поскольку одновременно выполняются две команды, процессор формирует два результата за цикл ($V_4[i]$ и $V_2[j]$) вместо одного. Без сцепления векторов пиковая производительность Cray-1 была бы 80 MFLOPS (один полный конвейер производит результат каждые 12,5 нс). При сцеплении трех конвейеров теоретический пик производительности - 240 MFLOPS. В принципе сцепление векторов можно

реализовать и в векторных процессорах типа «память-память», но для этого необходимо повысить пропускную способность памяти. Без сцепления необходимы три «канала»: два для входных потоков операндов и один — для потока результата. При использовании сцепления требуется обеспечить пять каналов: три входных и два выходных.

Завершая обсуждение векторных и векторно-конвейерных ВС, следует отметить, что с середины 90-х годов прошлого века этот вид ВС стал уступать свои позиции другим более технологичным видам систем. Тем не менее одна из последних разработок корпорации NEC (2002 год) - вычислительная система «Модель Земли» (The Earth Simulator), - являющаяся на сегодняшний момент самой производительной вычислительной системой в классе, по сути представляет собой векторно-конвейерную ВС. Система включает в себя 640 вычислительных узлов по 8 векторных процессоров в каждом. Пиковая производительность суперкомпьютера превышает 40 TFLOPS.

Матричные вычислительные системы

Назначение *матричных вычислительных систем* во многом схоже с назначением векторных ВС - обработка больших массивов данных. В основе матричных систем лежит *матричный процессор (*array processor), состоящий из регулярного массива процессорных элементов (ПЭ). Организация систем подобного типа на первый взгляд достаточно проста. Они имеют общее управляющее устройство, генерирующее поток команд, и большое число ПЭ, работающих параллельно и обрабатывающих каждый свой поток данных. Однако на практике, чтобы обеспечить достаточную эффективность системы при решении широкого круга задач, необходимо организовать связи между процессорными элементами так, чтобы наиболее полно загрузить процессоры работой. Именно характер связей между ПЭ и определяет разные свойства системы. Ранее уже отмечалось, что подобная схема применима и для векторных вычислений.

Между матричными и векторными системами есть существенная разница. Матричный процессор интегрирует множество идентичных функциональных блоков (ФБ), логически объединенных в матрицу и работающих в SIMD-стиле. Не столь существенно, как конструктивно реализована матрица процессорных элементов — на едином кристалле или на нескольких. Важен сам принцип - ФБ логически скомпонованы в матрицу и работают синхронно, то есть присутствует только один поток команд для всех. Векторный процессор имеет встроенные команды для обработки векторов данных, что позволяет эффективно загрузить конвейер из функциональных блоков, В свою очередь, векторные процессоры проще использовать, потому что команды для обработки векторов — это более удобная для человека модель программирования, чем SIMD.

Структуру матричной вычислительной системы можно представить в виде, показанном на рис. 13.10 [234].

Собственно параллельная обработка множественных элементов данных осуществляется *массивом процессоров* (МПр). Единый поток команд, управляющий обработкой данных в массиве процессоров, генерируется *контроллером массива*

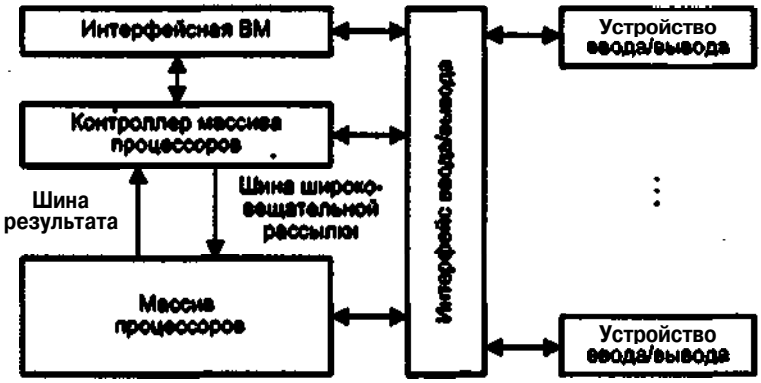


Рис. 13.10. Обобщенная модель матричной SIMD-системы

процессоров (КМП). КМП выполняет последовательный программный код, реализует операции условного и безусловного переходов, транслирует в МПр команды, данные и сигналы управления. Команды обрабатываются процессорами в режиме жесткой синхронизации. Сигналы управления используются для синхронизации команд и пересылок, а также для управления процессом вычислений, в частности определяют, какие процессоры массива должны выполнять операцию, а какие — нет. Команды, данные и сигналы управления передаются из КМП в массив процессоров по *шине широковещательной рассылки*. Поскольку выполнение операций условного перехода зависит от результатов вычислений, результаты обработки данных в массиве процессоров транслируются в КМП, проходя по *шине результата*.

Для обеспечения пользователя удобным интерфейсом при создании и отладке программ в состав подобных ВС обычно включают *интерфейсную ВМ* (front-end computer). В роли такой ВМ выступает универсальная вычислительная машина, на которую дополнительно возлагается задача загрузки программ и данных в КМП. Кроме того, загрузка программ и данных в КМП может производиться и напрямую *с устройств ввода/вывода,* например с магнитных дисков. После загрузки КМП приступает к выполнению программы, транслируя в МПр по широковещательной шине соответствующие SIMD-команды.

Рассматривая массив процессоров, следует учитывать, что для хранения множественных наборов данных в нем, помимо множества процессоров, должно присутствовать и множество модулей памяти. Кроме того, в массиве должна быть реализована сеть взаимосвязей, как между процессорами, так и между процессорами и модулями памяти. Таким образом, под термином *массив процессоров* понимают блок, состоящий из процессоров, модулей памяти и сети соединений.

Дополнительную гибкость при работе с рассматриваемой системой обеспечивает механизм *маскирования,* позволяющий привлекать к участию в операциях лишь определенное подмножество из входящих в массив процессоров. Маскирование реализуется как на стадии компиляции, так и на этапе выполнения, при этом процессоры, исключенные путем установки в ноль соответствующих битов маски, во время выполнения команды простаивают.

Интерфейсная ВМ

Интерфейсная ВМ (ИВМ) соединяет матричную SIMD-систему с внешним миром, используя для этого какой-либо из сетевых интерфейсов, например Ethernet, как это имеет место в системе MasPar MP-1. Интерфейсная ВМ работает под управлением операционной системы, чаще всего ОС UNIX. На ИВМ пользователи подготавливают, компилируют и отлаживают свои программы. В процессе выполнения программы сначала загружаются из интерфейсной ВМ в контроллер управления массивом процессоров, который выполняет программу и распределяет команды и данные по процессорным элементам массива. В некоторых ВС, например в Massively Parallel Computer MPP, при создании, компиляции и отладке программ КМП и интерфейсная ВМ используются совместно.

На роль ИВМ подходят различные вычислительные машины. Так, в системе СМ-2 в этом качестве выступает рабочая станция SUN-4, в системе MasPar — DECstation 3000, а в системе MPP - DEC VAX-11/780.

Контроллер массива процессоров

Контроллер массива процессоров выполняет последовательный программный код, реализует команды ветвления программы, транслирует команды и сигналы управления в процессорные элементы. Рисунок 13.11 иллюстрирует одну из возможных реализаций КМП, в частности принятую в устройстве управления системы PASM.

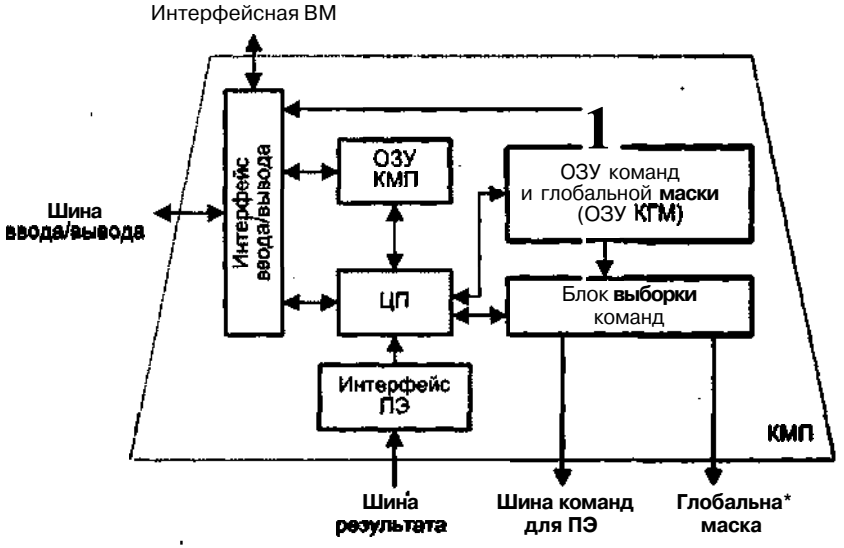


Рис. 13.11. Модель контроллера массива процессоров [225]

При загрузке из ИВМ программа через интерфейс ввода/вывода заносится в оперативное запоминающее устройство КМП (ОЗУ КМП). Команды для процессорных элементов и глобальная маска, формируемая на этапе компиляции, также через интерфейс ввода/вывода загружаются в ОЗУ команд и глобальной маски

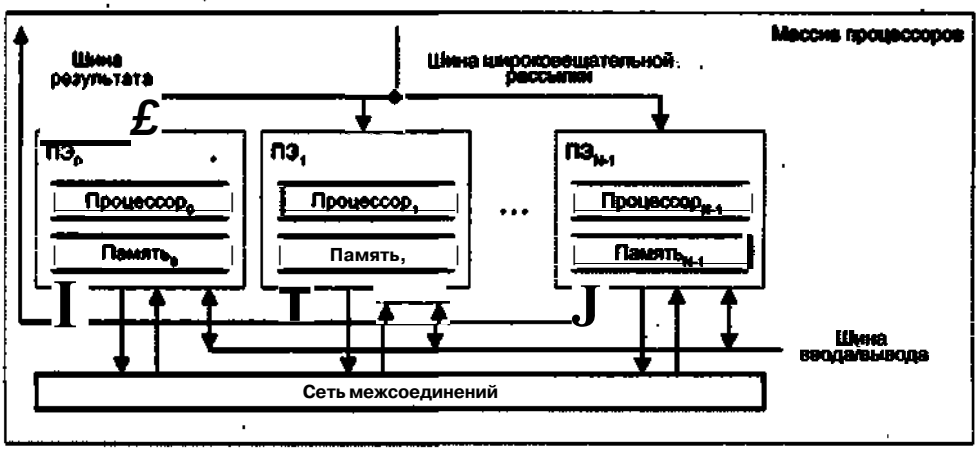
(ОЗУ КГМ). Затем КМП начинает выполнять программу, извлекая либо одну скалярную команду из ОЗУ КМП, либо множественные команды из ОЗУ КГМ. Скалярные команды - команды, осуществляющие операции над хранящимися в КМП скалярными данными, выполняются центральным процессором (ЦП) контроллера массива процессоров. В свою очередь, команды, оперирующие параллельными переменными, хранящимися в каждом ПЭ, преобразуются в блоке выборки команд в более простые единицы выполнения - *нанокоманды*. Нанокоманды совместно с маской пересылаются через шину команд для ПЭ на исполнение в массив процессоров. Например, команда сложения 32-разрядных слов в КМП системы MPP преобразуется в 32 нанокоманды одноразрядного сложения, которые каждым ПЭ обрабатываются последовательно,

В большинстве алгоритмов дальнейший порядок вычислений зависит от результатов и/или флагов условий предшествующих операций. Для обеспечения такого режима в матричных системах статусная информация, хранящаяся в процессорных элементах, должна быть собрана в единое слово и передана в КМП для выработки решения о ветвлении программы. Например, в предложении IF ALL (условие A) THEN DO B оператор B будет выполнен, если условие A справедливо во всех ПЭ. Для корректного включения/отключения процессорных элементов КМП должен знать результат проверки условия A во всех ПЭ. Такая информация передается в КМП по однонаправленной шине результата. В системе CM-2 эта шина названа GLOBAL. В системе MPP для той же цели организована структура, называемая *деревом SUM-OR*. Каждый ПЭ помещает содержимое своего одноразрядного регистра признака на входы дерева, которое с помощью операции логического сложения комбинирует эту информацию и формирует слово результата, используемое в КМП для принятия решения.

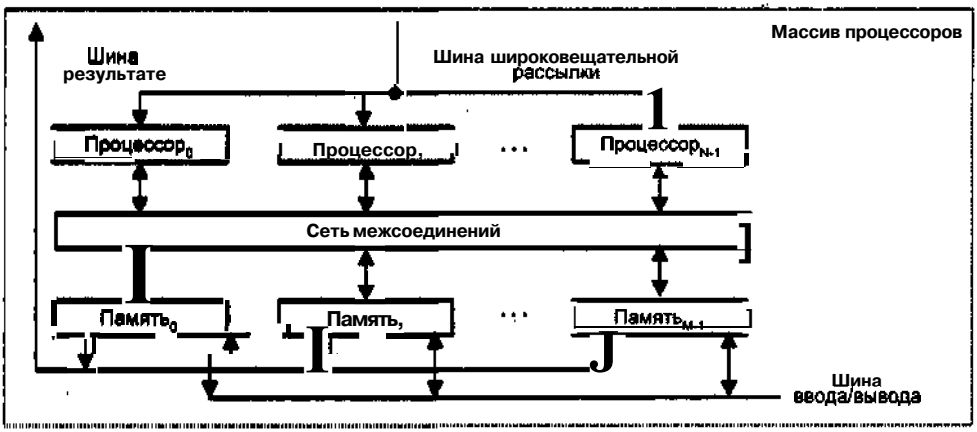
Массив процессоров

В матричных SIMD-системах распространение получили два основных типа архитектурной организации массива процессорных элементов (рис. 13.12).

В первом варианте, известном как архитектура типа *«процессорный элемент-процессорный элемент»* («ПЭ-ПЭ»), *N* процессорных элементов (ПЭ) связаны между собой сетью соединений (рис, 13.12, а). Каждый ПЭ - это процессор с локальной памятью. Процессорные элементы выполняют команды, получаемые из КМП по шине широковещательной рассылки, и обрабатывают данные как хранящиеся в их локальной памяти, так и поступающие из КМП, Обмен данными между процессорными элементами производится по *сети соединений,* в то время как *шина ввода/вывода* служит для обмена информацией между ПЭ и устройствами ввода/вывода. Для трансляции результатов из отдельных ПЭ в контроллер массива процессоров служит *шина результата*. Благодаря использованию локальной памяти аппаратные средства ВС рассматриваемого типа могут быть построены весьма эффективно. Во многих алгоритмах действия по пересылке информации по большей части локальны, то есть происходят между ближайшими соседями. По этой причине архитектура, где каждый ПЭ связан только с соседними, очень популярна. В качестве примеров вычислительных систем с рассматриваемой архитектурой можно упомянуть MasPar MP-1, Connection Machine CM-2, GF11, DAP, MPP, STARAN, PEPE, ILLIAC IV.



а



б

Рис. 13.12. Модели массивов процессоров: а - «процессорный элемент-процессорный элемент»; *б* - « процессор-память»

Второй вид архитектуры — *«процессор-память»показ*ан на рис, 13.12,*б.* В такой конфигурации двунаправленная сеть соединений связывает *N* процессоров с *M* модулями памяти. Процессоры управляются КМП через широковещательную шину. Обмен данными между процессорами осуществляется как через сеть, так и через модули памяти. Пересылка данных между модулями памяти и устройствами ввода/вывода обеспечивается шиной ввода/вывода. Для передачи данных из конкретного модуля памяти в КМП служит шина результата. Примерами ВС с рассмотренной архитектурой могут служить Burroughs Scientific Processor (BSP), Texas Reconfigurable Array Computer TRAC,

Структура процессорного элемента

В большинстве матричных SIMD-систем в качестве процессорных элементов применяются простые RISC-процессоры с локальной памятью ограниченной емкости.

Например, каждый ПЭ системы MasPar MP-1 состоит из четырехразрядного процессора с памятью емкостью 64 Кбайт. В системе MPP используются одноразрядные процессоры с памятью 1 кбит каждый, а в СМ-2 процессорный элемент представляет собой одноразрядный процессор с 64 Кбит локальной памяти. Благодаря простоте ПЭ массив может быть реализован в виде одной сверхбольшой интегральной микросхемы (СБИС). Это позволяет сократить число связей между микросхемами и, следовательно, габариты ВС. Так, одна СБИС в системе СМ-2 содержит 16 процессоров (без блоков памяти), а в системе MasPar MP-1 СБИС состоит из 32 процессоров (также без блоков памяти). В системе MP-2 просматривается тенденция к применению более сложных микросхем, в частности 32-разрядных процессоров с 256 Кбайт памяти в каждом.

Неотъемлемыми компонентами ПЭ (рис. 13,13) в большинстве вычислительных систем являются:

- арифметико-логическое устройство (АЛУ);
- регистры данных;
- сетевой интерфейс (СИ), который может включать в свой состав регистры пересылки данных;
- номер процессора;
- регистр флага разрешения маскирования (F);
- локальная память.

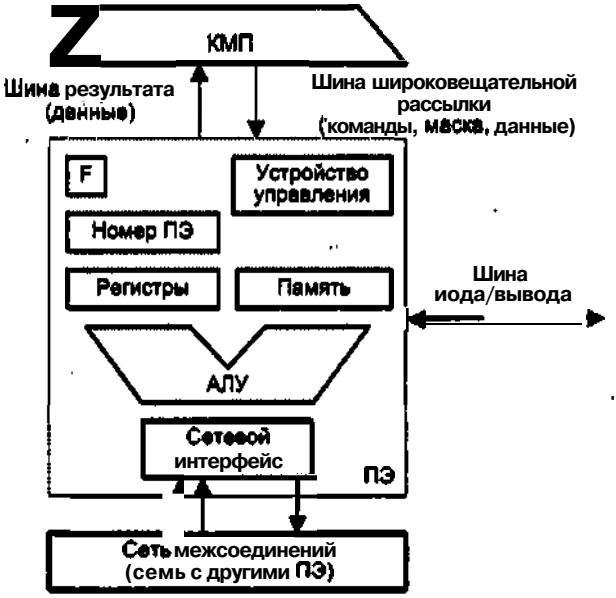


Рис. 13.13. Модель процессорного элемента [225]

Процессорные элементы, управляемые командами, поступающими по широковещательной шине из КМП, могут выбирать данные из своей локальной памяти и регистров, обрабатывать их в АЛУ и сохранять результаты в регистрах и локаль-

ной памяти. ПЭ могут также обрабатывать те данные, которые поступают по шине широковещательной рассылки из КМП. Кроме того, каждый процессорный элемент вправе получать данные из других ПЭ и отправлять их в другие ПЭ по сети соединений, используя для этого свой сетевой интерфейс. В некоторых матричных системах, в частности в MasPar MP-1, элемент данных из ПЭ-источника можно передавать в ПЭ-приемник непосредственно, в то время как в других, например в MPP, - данные предварительно должны быть помещены в специальный регистр пересылки данных, входящий в состав сетевого интерфейса. Пересылка данных между ПЭ и устройствами ввода/вывода осуществляется через шину ввода/вывода ВС. В ряде систем (MasPar MP-1) ПЭ подключены к шине ввода/вывода посредством сети соединений и канала ввода/вывода системы. Результаты вычислений любое ПЭ выдает в КМП через шину результата.

Каждому из N ПЭ в массиве процессоров присваивается уникальный номер, называемый *также адресом ПЭ,* который представляет собой целое число от 0 до *N*- 1. Чтобы указать, должен ли данный ПЭ участвовать в общей операции, него составе имеется регистр флага разрешения F. Состояние этого регистра определяют сигналы управления из КМП, либо результаты операций в самом ПЭ, либо и те и другие совместно.

Еще одной существенной характеристикой матричной системы является способ синхронизации работы ПЭ. Так как все ПЭ получают и выполняют команды одновременно, их работа жестко синхронизируется. Это особенно важно в операциях пересылки информации между ПЭ. В системах, где обмен производится с четырьмя соседними ПЭ, передача информации осуществляется в режиме «регистр-регистр».

Подключение и отключение процессорных элементов

В процессе вычислений в ряде операций должны участвовать только определенные ПЭ, в то время как остальные ПЭ остаются бездействующими. Разрешение и запрет работы ПЭ могут исходить от контроллера массива процессоров *(глобальное маскирование)* и реализуются с помощью схем маскирования ПЭ. В этом случае решение о необходимости маскирования принимается на этапе компиляции кода Решение о маскировании может также приниматься во время выполнения программы *(маскирование, определяемое данными),* при этом опираются на хранящийся в ПЭ флаг разрешения маскирования F.

При маскировании, определяемом данными, каждый ПЭ самостоятельно объявляет свой статус «подключен/не подключен». В составе системы команд имеются наборы маскируемых и немаскируемых команд. Маскируемые команды выполняются в зависимости от состояния флага F, в то время как немаскируемые флаг просто игнорируют. Процедуру маскирования рассмотрим на примере предложения IFTHEN-ELSE, Пусть *x* - локальная переменная (хранящаяся в локальной памяти каждого ПЭ). Предположим, что процессорные элементы массива параллельно выполняют, ветвление:

IF (x > 0) THEN <оператор A> ELSE <оператор B>

и каждый ПЭ оценивает условие IF. Те ПЭ, для которых условие $x > 0$ справедливо, установят свой флаг F в единицу, тогда как остальные ПЭ - в ноль. Далее КМ П

распределяет оператор A по всем ПЭ. Команды, реализующие этот оператор, должны быть маскируемыми. Оператор A будет выполнен только теми ПЭ, где флаг F установлен в единицу. Далее КМП передает во все ПЭ немаскируемую команду ELSE, которая заставит все ПЭ инвертировать состояние своего флага F. Затем КМП транслирует во все ПЭ оператор B, который также должен состоять из маскируемых команд. Оператор будет выполнен теми ПЭ, где флаг F после инвертирования был установлен в единицу, то есть где результат проверки условия $x > 0$ был отрицательным.

При использовании схемы глобального маскирования контроллер массива процессоров вместе с командами посылает во все ПЭ глобальную маску. Каждый ПЭ декодирует эту маску и по результату выясняет, должен ли он выполнять данную команду или нет.

В зависимости от способа кодирования маски существует несколько различных схем глобального маскирования. В схеме, примененной в вычислительной системе ILLIAC IV с 64 64-разрядными ПЭ, маска представляет собой N-разрядный вектор. Каждый бит вектора отражает состояние одного ПЭ. Если бит содержит единицу, соответствующий ПЭ будет активным, в противном случае — пассивным. Несмотря на свою универсальность, при больших значениях *N* схема становится неудобной. В варианте маскирования с адресом ПЭ используется *2m*-разрядная маска *(m* $= \log_2 N$), в которой каждая позиция соответствует одному разряду в двоичном представлении адреса ПЭ. Каждая позиция может содержать 0, i или *X.* Таким образом, маска состоит из *2m* битов. Если для всех *i* $(0 <= i < m)$ *i-я* позиция в маске и i-я позиция в адресе ПЭ совпадают или в i-й позиции маски стоит *X,* ПЭ будет активным. Например, маска 000X1 представляет процессорные элементы с номерами 1 и 3, в то время как маска XXXXO представляет все ПЭ с четными номерами (все это для массива из 32 ПЭ). Здесь можно активизировать только подмножество из всех возможных комбинаций процессорных элементов массива, что на практике не является ограничением, так как в реальных алгоритмах обычно участвуют не произвольные ПЭ, а лишь расположенные регулярным образом.

Глобальные и локальные схемы маскирования могут комбинироваться. В таком случае активность ПЭ в равной мере определяется как флагом F, так и глобальной маской.

Сети взаимосвязей процессорных элементов

Эффективность сетей взаимосвязей процессорных элементов во многом определяет возможную производительность всей матричной системы. Применение находят самые разнообразные топологии сетей.

Поскольку процессорные элементы в матричных системах функционируют синхронно, обмениваться информацией они также должны по согласованной схеме, причем необходимо обеспечить возможность синхронной передачи от нескольких ПЭ-источников к одному ПЭ-приемнику. Когда для передачи информации в сетевом интерфейсе задействуется только один регистр пересылки данных, это может привести к потере данных, поэтому в ряде ВС для предотвращения подобной ситуации предусмотрены специальные механизмы. Так, в системе СМ-2 используется оборудование, объединяющее сообщения, поступившие к одному ПЭ.

Объединение реализуется за счет операций арифметического и логического сложения, наложения записей, нахождения меньшего и большего из двух значений. В некоторых SIMD-системах, например MP-1, имеется возможность записать одновременно пришедшие сообщения в разные ячейки локальной памяти.

Хотя пересылки данных по сети инициируются только активными ПЭ, пассивные процессорные элементы также вносят вклад в эти операции. Если активный ПЭ инициирует чтение из другого ПЭ, операция выполняется вне зависимости от статуса ПЭ, из которого считывается информация. То же самое происходит и при записи.

Наиболее распространенными топологиями в матричных системах являются решетчатые и гиперкубические. Так, в ILLIAC IV, MPP и CM-2 каждый ПЭ соединен с четырьмя соседними. В MP-1 и MP-2 каждый ПЭ связан с восьмью смежными ПЭ. В ряде систем реализуются многоступенчатые динамические сети соединений (MP-1, MP-2, GF11).

Ввод/вывод

Хотя программа вычислений хранится в памяти интерфейсной ВМ или иногда в КМП, входные и выходные данные процессорных элементов и КМП могут храниться также на внешних ЗУ. Такие ЗУ могут подключаться к массиву процессоров и/или КМП посредством каналов ввода/вывода или процессоров ввода/вывода.

Ассоциативные вычислительные системы

К классу SIMD относятся и так называемые ассоциативные вычислительные системы. В основе подобной ВС лежит ассоциативное запоминающее устройство (см. главу 5), а точнее - *ассоциативный процессор,* построенный на базе такого ЗУ. Напомним, что ассоциативная память (или ассоциативная матрица) представляет собой ЗУ, где выборка информации осуществляется не по адресу операнда, а по отличительным признакам операнда. Запись в традиционное ассоциативное ЗУ также производится не по адресу, а в одну из незанятых ячеек.

Ассоциативный процессор (АП) можно определить как ассоциативную память, допускающую параллельную запись во все ячейки, для которых было зафиксировано совпадение с ассоциативным признаком. Эта особенность АП, носящая название *мультизаписи,* является первым отличием ассоциативного процессора от традиционной ассоциативной памяти. Считывание и запись информации могут производиться по двум срезам запоминающего массива — либо это все разряды одного слова, либо один и тот же разряд всех слов. При необходимости выделения отдельных разрядов среза лишние позиции допустимо маскировать. Каждый разряд среза в АП снабжен собственным процессорным элементом, что позволяет между считыванием информации и ее записью производить необходимую обработку, то есть параллельно выполнять операции арифметического сложения, поиска, а также эмулировать многие черты матричных ВС, таких, например, как ILLIAC IV.

Таким образом, *ассоциативные ВС* или *ВС с ассоциативным процессором* есть не что иное, как одна из разновидностей параллельных ВС, в которых *n* процес-

сорных элементов ПЭ (вертикальный разрядный срез памяти) представляют собой простые устройства, как правило, последовательной поразрядной обработки. При этом каждое слово (ячейка) ассоциативной памяти имеет свое собственное устройство обработки данных (сумматор). Операция осуществляется одновременно всеми *n* ПЭ. Все или часть элементарных последовательных ПЭ могут синхронно выполнять операции над всеми ячейками или над выбранным множеством слов ассоциативной памяти.

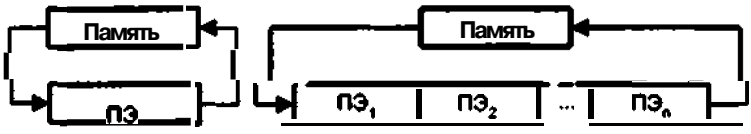
Время обработки *N* m-разрядных слов в ассоциативной ВС определяется выражением:

$$T = m \times t \times \left(\frac{N}{n} + K \right),$$

где *t* - время цикла ассоциативной памяти; *n* - число ячеек ассоциативной системы; *K* — коэффициент сложности выполнения элементарной операции (количество последовательных шагов, каждый из которых связан с доступом к памяти).

Вычислительные системы с систолической структурой

В фон-неймановских машинах данные, считанные из памяти, однократно обрабатываются в процессорном элементе, после чего снова возвращаются в память (рис. 13.14 *а)*. Авторы идеи систолической матрицы Кунг и Лейзерсон предложили организовать вычисления так, чтобы данные на своем пути от считывания из памяти до возвращения обратно пропускались через как можно большее число ПЭ (рис. 13.14,6).



а б

Рис. 13.14. Обработка данных в ВС: а - фон-неймановского типа; *б* - систолической структуры

Если сравнить положение памяти в ВС со структурой живого организма, то по аналогии ей можно отвести роль сердца, множеству ПЭ — роль тканей, а поток данных рассматривать как циркулирующую кровь. Отсюда и происходит название *систолическая матрица* (систола - сокращение предсердий и желудочков сердца при котором кровь нагнетается в артерии). Систолические структуры эффективны при выполнении матричных вычислений, обработке сигналов, сортировке данных и т. д. В качестве примера авторами идеи был предложен линейный массив для алгоритма матричного умножения, показанный на рис. 13.15.

В основе схемы лежит ритмическое прохождение двух потоков данных x_i и y, навстречу друг другу. Последовательные элементы каждого потока разделены одним тактовым периодом, чтобы любой из них мог встретиться с любым элементом встречного потока. Если бы они следовали в каждом периоде, то элемент x_i никогда

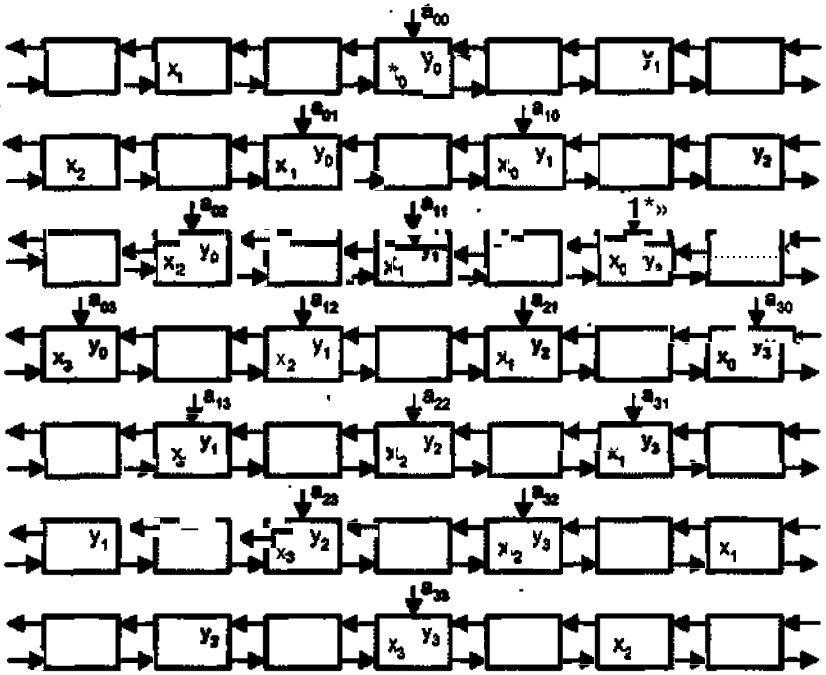


Рис. 13.15. Процесс векторного умножения матриц (n = 4)

бы не встретился с элементамиу y_{i+1}, y_{i+3}... Вычисления выполняются параллельно в процессорных элементах, каждый из которых реализует один шаг в операции вычисления скалярного произведения (IPS, Inner Product Step) и носит название *IPS-элемента* (рис. 13.16).

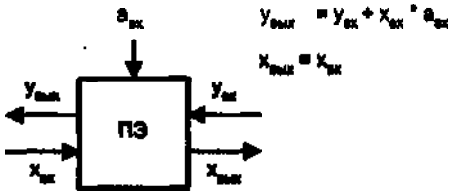


Рис. 13.16. Функциональная схема IPS-элемента

Значение $y_{вх}$, поступающее на вход ПЭ, суммируется с произведением входных значений $x_{вх}$ и $a_{вх}$. Результат выходит из ПЭ как $y_{ВЫХ}$ Значение $x_{вх}$, кроме того, для возможного последующего использования остальной частью массива транслируется через ПЭ без изменений и покидает его в виде $x_{вых}$.

Таким образом, *систолическая структура* — это однородная вычислительная среда из процессорных элементов, совмещающая в себе свойства конвейерной и матричной обработки и обладающая следующими особенностями:

- вычислительный процесс в систолических структурах представляет собой непрерывную и регулярную передачу данных от одного ПЭ к другому без запоминания промежуточных результатов вычисления;

- каждый элемент входных данных выбирается из памяти однократно и используется столько раз, сколько необходимо по алгоритму, ввод данных осуществляется в крайние ПЭ матрицы;
- образующие систолическую структуру ПЭ однотипны и каждый из них может быть менее универсальным,чем процессоры обычных многопроцессорных систем;
- потоки данных и управляющих сигналов обладают регулярностью, что позволяет объединять ПЭ локальными связями минимальной длины;
- алгоритмы функционирования позволяют совместить параллелизм с конвейерной обработкой данных;
- производительность матрицы можно улучшить за счет добавления в нее определенного числа ПЭ, причем коэффициент повышения производительности при этом линеен.

В настоящее время достигнута производительность систолических процессоров порядка 1000 млрд операций/с.

Классификация систолических структур

Анализ различных типов систолических структур и тенденций их развития позволяет классифицировать эти структуры по нескольким признакам.

По *степени гибкости* систолические структуры могут быть сгруппированы на

- специализированные;
- алгоритмически ориентированные;
- программируемые.

Специализированные структуры ориентированы на выполнение определенного алгоритма. Эта ориентация отражается не только в конкретной геометрии систолической структуры, статичности связей между ПЭ и числе ПЭ, но и в выборе типа операции, выполняемой всеми ПЭ. Примерами являются структуры, ориентированные на рекурсивную фильтрацию, быстрое преобразование Фурье для заданного количества точек, конкретные матричные преобразования.

Алгоритмически ориентированные структуры обладают возможностью программирования либо конфигурации связей в систолической матрице, либо самих ПЭ. Возможность программирования позволяет выполнять на таких структурах некоторое множество алгоритмов, сводимых к однотипным операциям над векторами, матрицами и другими числовыми множествами.

В программируемых систолических структурах имеется возможность программирования как самих ПЭ, так и конфигурации связей между ними. При этом ПЭ могут обладать локальной памятью программ, и хотя все они имеют одну и ту же организацию, в один и тот же момент времени допускается выполнение различных операций из некоторого набора. Команды или управляющие слова, хранящиеся в памяти программ таких ПЭ, могут изменять и направление передачи операндов.

По разрядности процессорных элементов систолические *структуры* делятся

- одноразрядные;
- многоразрядные.

В одноразрядных матрицах ПЭ в каждый момент времени выполняет операцию над одним двоичным разрядом; а в многоразрядных — над словами фиксированной длины.

По характеру локально-пространственных связей систолические структуры бывают:

- одномерные;
- двухмерные;
- трехмерные.

Выбор структуры зависит от вида обрабатываемой информации. Одномерные схемы применяются при обработке векторов, двухмерные - матриц, трехмерные - множеств иного типа.

Топология систолических структур

В настоящее время разработаны систолические матрицы с различной геометрией связей: линейные, квадратные, гексагональные, Трехмерные и др. Перечисленные конфигурации систолических матриц приведены на рис. 13.17.

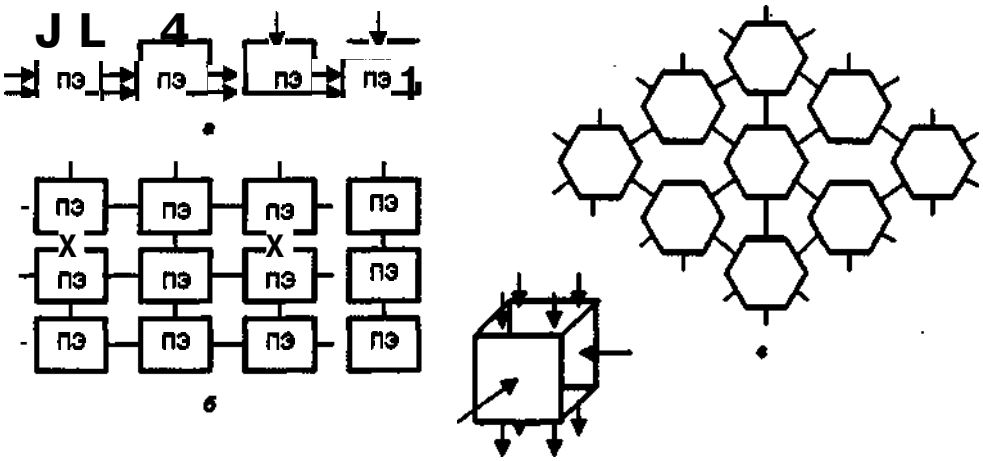


Рис. 13.17. Конфигурация систолических матриц: а - линейная; б- прямоугольная; *в* — гексагональная; г — трехмерная

Каждая конфигурация матрицы наиболее приспособлена для выполнения определенных функций, например линейная матрица оптимальна для реализации фильтров в реальном масштабе времени; гексагональная — для выполнения операций обращения матриц, а также действий над матрицами специального вида (Теплица- Генкеля); трехмерная - для нахождения значений нелинейных дифференциальных уравнений в частных производных или для обработки сигналов антенной решетки. Наиболее универсальными и наиболее распространенными, тем не менее, можно считать матрицы с линейной структурой.

Для решения сложных задач конфигурация систолической структуры может представлять собой набор отдельных матриц, сложную сеть взаимосвязанных мат-

риц либо обрабатывающую поверхность. *Под обрабатывающей поверхностью понимается* бесконечная прямоугольная сетка ПЭ, где каждый ПЭ соединяется со своими четырьмя соседями (или большим числом ПЭ). Одним из наиболее подходящих элементов для реализации обрабатывающей поверхности является матрица простых ПЭ или транспьютеров.

Учитывая то, что матрицы ПЭ обычно реализуются на основе сверхбольших интегральных схем, возникающие при этом ограничения привели к тому, что наиболее распространены матрицы с одним, двумя и тремя трактами данных и с одинаковым либо противоположным направлением передачи, обозначаемые как ULA, BLA и TLA соответственно.

ULA (Unidirectional Linear Array) - это однонаправленный линейный процессорный массив, где потоки данных перемещаются в одном направлении. ПЭ в массиве могут быть связаны одним, двумя или тремя трактами.

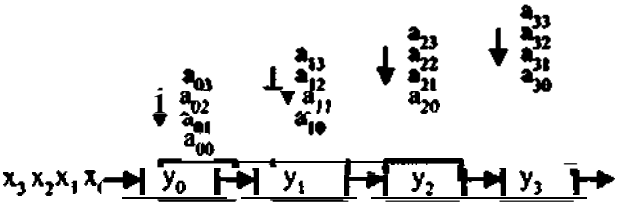


Рис. 13.18. Поток данных при векторном перемножении матрице ULA (n = 4)

При реализации алгоритма векторного произведения матриц один из потоков данных перемещается вправо, в то время как второй резидентно расположен в массиве (рис. 13.18). Используемый ПЭ представляет собой модифицированный IPS-элемент, поскольку имеется только один тракт данных, а элементы второго потока хранятся в ПЭ массива.

BLA (Bidirectional Linear Array) - это двунаправленный линейный процессорный массив, в котором два потока данных движутся навстречу друг другу. BLA, где один из потоков является выходным, называется *регулярным*.

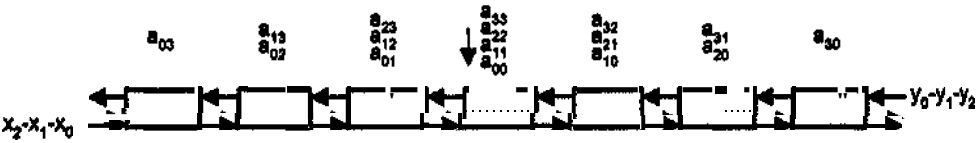


Рис. 13.19. Поток данных при векторном перемножении матриц в BLA (n = 4)

Реализация рассмотренной ранее операции с применением BLA показана на рис. 13.19. В версии ULA процессоры используются более эффективно, поскольку в них элементы потока следуют в каждом такте, а не через такт, как в BLA

TLA (Three-path communication Linear Array) - линейный процессорный массив с тремя коммуникационными трактами, в котором по разным направлениям перемещаются три потока данных. На рис. 13.20 показан пример фильтра ARMA, предложенного Кунгом и построенного по схеме TLA Возможны несколько вариантов такого фильтра, в зависимости от числа выходных потоков данных и от зна-

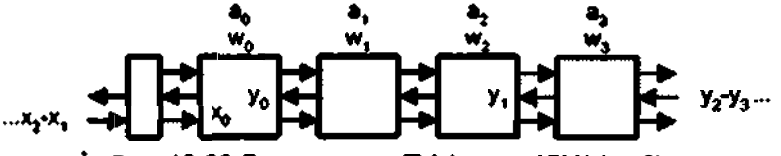


Рис. 13.20. Поток данных в TLA фильтра ARMA (n = 3)

чений, хранящихся в памяти (в примере фигурирует один выходной поток). Процессорные элементы выполняют две операции IPS и обычно называются *сдвоенными IPS-элементами,* Две версии таких ПЭ представлены на рис, 13.21. ПЭ могут использовать как хранимые в памяти значения (рис. 13.21, *а, б),* так и внешние данные (рис. 13.21, *в, г).*

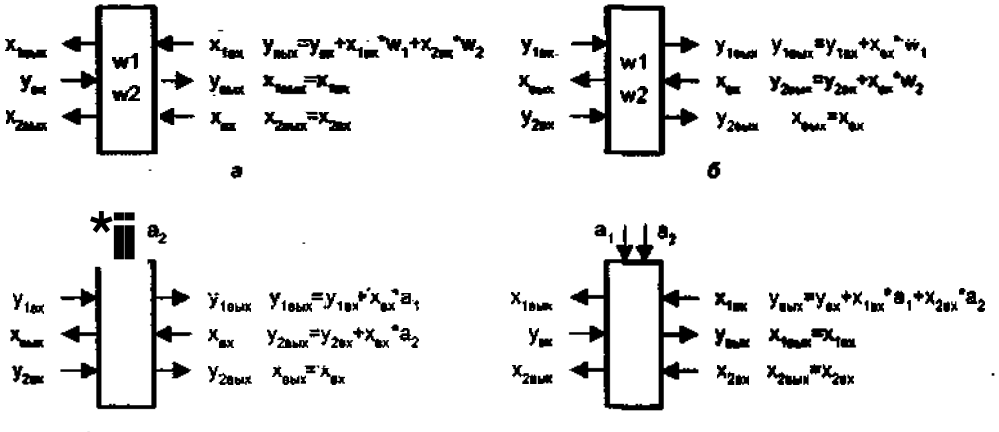


Рис. 13.21. Сдвоенные IP S-элементы:а-*б* — с хранимыми в памяти двумя значениями; *в-г* — с внешними данными

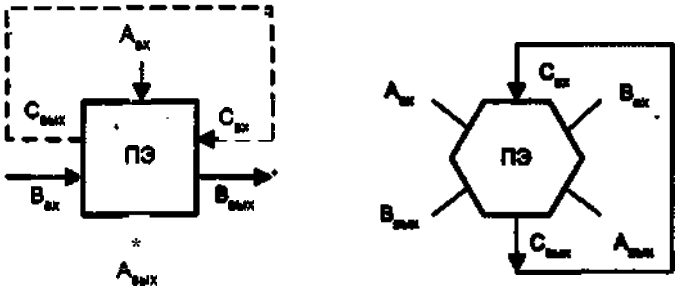
TLA часто называют сдвоенным конвейером, поскольку он может быть разделен на два линейных конвейера типа BLA Соответственно, TLA можно получить объединением двух BLA с одним общим потоком данных.

Представленные реализации алгоритма векторного произведения матриц выполняют эту операцию за одно и то же время, но в случае ULA в вычислениях участвуют вдвое меньше процессорных элементов. С другой стороны, ULA использует хранимые в памяти данные, на чтение и запись которых нужно какое-то время. В свою очередь, в схеме BLA требуется дополнительное время на операции ввода/вывода.

Структура процессорных элементов

Тип ПЭ выбирается в соответствии с назначением систолической матрицы и структурой пространственных связей. Наиболее распространены процессорные элементы, ориентированные на умножение с накоплением.

На рис. 13.22 показаны ПЭ для двух типов матриц: прямоугольной (см. рис, 13.22, *а)* и гексагональной (см. рис. *13.22,б).*



а б

Рис. 13.22. Структура ПЭ: а - для прямоугольной систолической матрицы; *б* - для гексагональной систолической матрицы

В обоих случаях на вход ПЭ подаются два операнда $A_{вх}$, $B_{вх}$, а выходят операнды $A_{вых}$, $B_{вых}$, и частичная сумма $C_{вых}$,. На n-м шаге работы систолической системы ПЭ выполняет операцию

$$C_{вых}^{(n)} = C_{вх}^{(n-1)} + A_{вх}^{(n-1)} \cdot B_{вх}^{(n-1)}$$

на основе операндов, полученных на (n - 1)-м шаге, при этом операнды на входе и выходе ПЭ одинаковы:

$$A_{вых}^{(n)} = A_{вх}^{(n-1)}, \; B_{вых}^{(n)} = B_{вх}^{(n-1)}.$$

Частичная сумма поступает на вход ПЭ либо с данного процессорного элемента (штриховая линия), либо с соседнего ПЭ матрицы.

Пример вычислений с помощью систолического процессора

Организацию вычислительного процесса в систолических массивах различной конфигурации с использованием ПЭ, функциональная схема которого показана на рис. 1.3.23. удобнее всего пояснить на примере умножения матрицы A -(a_y)на вектор **X** - $\{x_1, x_2, ..., x_n\}$.

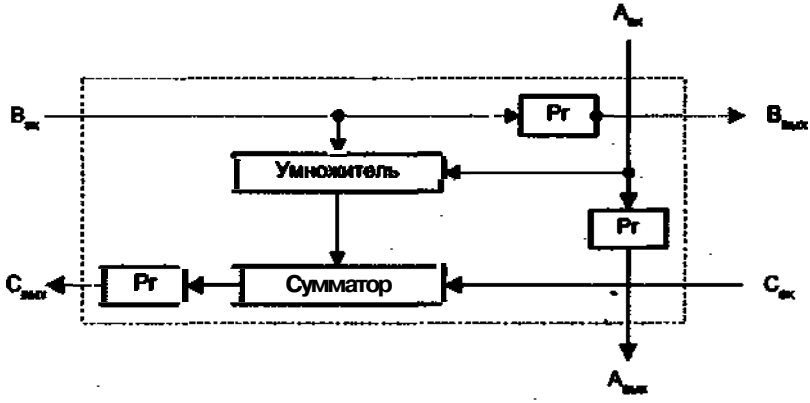


Рис. 13.23. Функциональная схема процессорного Элемента систолической матрицы

Элементы вектора произведения $Y = \{y_1, y_2, \ldots, y_n\}$ могут быть получены периодически повторяющимися операциями

$$y_i(1) = 0; y_i(k+1) = y_i(k) + a_i(k) \times x_i(k); y_i = y_i(n+1),$$

где k - номер шага вычислений.

Пусть имеется матрица А размером nxn с шириной полосы ненулевых элементов $p + q - 1 = 4$. Схема умножения вектора на матрицу в этом случае представлена на рис. 13.24.

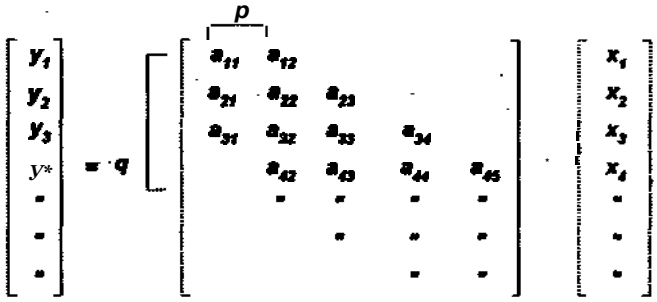


Рис. 13.24. Схема умножения вектора на матрицу

Определенная выше последовательность операций для вычисления компонентов вектора Y может быть получена за счет конвейерного прохождения x_i и y_i через $p + q - 1$ последовательно соединенных ПЭ (рис. 13.25)

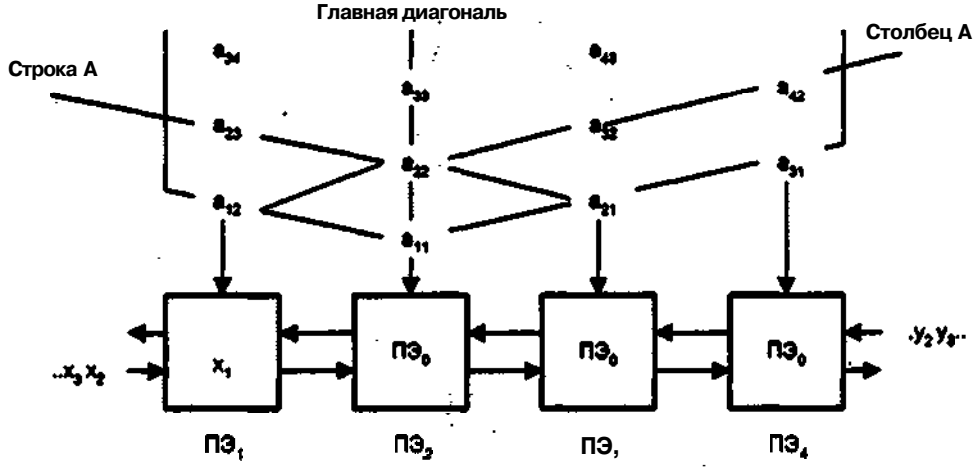


Рис. 13.25. Организация вычислений в линейной систолической структуре

Компоненты y_i (i - 1,...,n) вектора Y; имеющие в начальный момент нулевое значение, поступают на вход массива и продвигаются через ПЭ справа налево, в то время как компоненты вектора X движутся слева направо. Элементы матрицы $A = \{a_{ij}\}$ в порядке, указанном на рисунке, вводятся в ПЭ сверху вниз. Промежуточные результаты $y_i(k)$ накапливаются по мере продвижения от одного ПЭ к другому.

В табл. 13.1 показаны первые 6 шаГов алгоритма умножения для рассматриваемой структуры.

Таблица 13.1. Последовательность умножения матрицы на вектор в систолической ВС

Шаг	Состояние				Комментарий
	$ПЭ_1$	$ПЭ_2$	$ПЭ_3$	$ПЭ_4$	
0				ft	Элемент y, поступил в ПЭ,
1	x_1		ft		Элемент x_1 поступил в $ПЭ_1$, элемент y_1 движется влево
2		ft «II x_1		ft	Элементе,, поступил в $ПЭ_2$, y_1 - у, + л,, $\times x_1$, то есть $y_1 = a_{11} + x_1$
3	ft a_{12} x_2		ft a_{21} x_1		Элемент a_{12} поступил в $ПЭ_1$, a_{21} — в $ПЭ_3$, ft'-«НХ-t.⁺«.jX^ft-«;.^^
4		ft a_{22} x_2		ft a_{31} x_1	Элемент y_1 вышел из $ПЭ_1$, $y_2 = a_{21} \times x_1 + a_{22} \times x_2$, ft" $a_{31} \times x_1$
5	ft a_{23} x_3		ft a_{32} x_2		$y_2 = a_{21} \times x_1 + a_{22} \times x_2 + a_{23} \times x_3$, $y_3 = a_{31} \times x_1 + a_{32} \times x_2$
6		ft a_{13} x_3		ft a_{42} x_2	Элемент y_2 вышел из $ПЭ_1$, $y_4 = a_{42} \times x_2$, ft $= a_{31} \times x_1 + a_{32} \times x_2 + a_{33} \times x_3$

Заметим, что при такой организации вычислительного процесса для каждого ПЭ такты выполнения операции чередуются с тактами простоя. Таким образом, в каждый момент времени активны только (p+q-1)/2 процессорных элементов, следовательно, каждый выходной результат формируется за два такта. Для вычисления всех и элементов выходного вектора Y необходимо $2n + p + q - 1$ тактов.

Вычислительные системы с командными словами сверхбольшой длины (VLIW)

Архитектура с командными словами сверхбольшой длины или *со сверхдлинными командами* (VLIW, Very Long Instruction Word) известна с начала 80-х из ряда университетских проектов, но только сейчас, с развитием технологии производства микросхем она нашла свое достойное воплощение. VLIW - это набор команд, организованных наподобие горизонтальной микрокоманды в микропрограммном устройстве управления.

Идея VLIW базируется на том, что задача эффективного планирования параллельного выполнения нескольких команд возлагается на «разумный» компилятор. Такой компилятор вначале исследует исходную программу с целью обнаружить все команды, которые могут быть выполнены одновременно, причем так,

чтобы это не приводило к возникновению конфликтов. В процессе анализа компилятор может даже частично имитировать выполнение рассматриваемой программы. Наследующем этапе компилятор пытается объединить такие команды в пакеты, каждый из которых рассматривается так одна сверхдлинная команда. Объединение нескольких простых команд в одну сверхдлинную производится по следующим правилам:

- количество простых команд, объединяемых в одну команду сверхбольшой длины, равно числу имеющихся в процессоре функциональных (исполнительных) блоков (ФБ);
- в сверхдлинную команду входят только такие простые команды, которые исполняются разными ФБ, то есть обеспечивается одновременное исполнение всех составляющих сверхдлинной команды.

Длина сверхдлинной команды обычно составляет от 256 до 1024 бит. Такая *метакоманда* содержит несколько полей (по числу образующих ее простых команд), каждое из которых описывает операцию для конкретного функционального блока. Сказанное иллюстрирует рис, 13,26, где показан возможный формат сверхдлинной команды и взаимосвязь между ее полями и ФБ, реализующими отдельные операции.

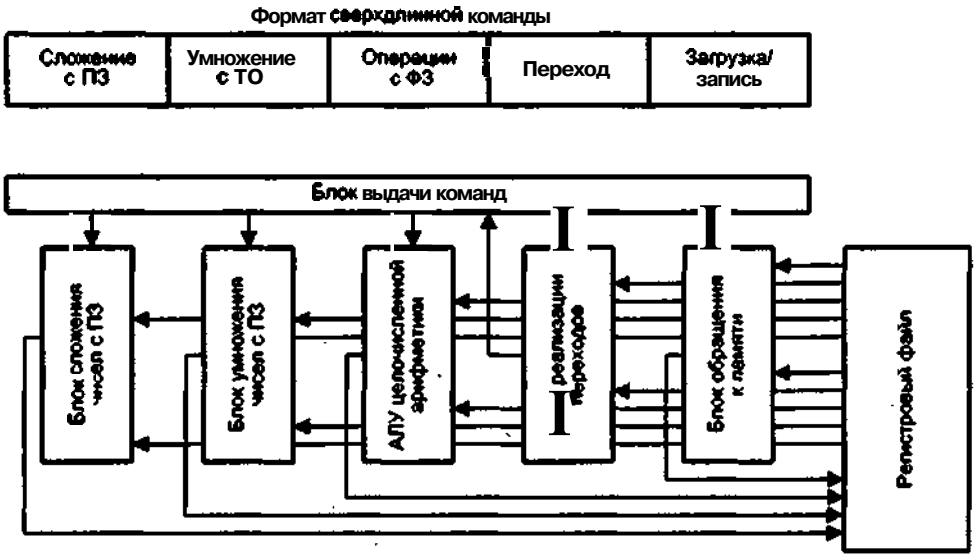


Рис. 13.26. формат сверхдлинной команды и взаимосвязь полей команды с составляющими блока исполнения

Как видно из рисунка, каждое поле сверхдлинной команды отображается на свой функциональный блок, что позволяет получить максимальную отдачу от аппаратуры блока исполнения команд.

VLIW-архитектуру можно рассматривать как статическую суперскалярную архитектуру. Имеется в виду, что распараллеливание кода производится на этапе компиляции, а не динамически во время исполнения. То, что в выполняемой сверх-

длинной команде исключена возможность конфликтов, позволяет предельно упростить аппаратуру VLIW-процессора и, как следствие, добиться более высокого быстродействия.

В качестве простых команд, образующих сверхдлинную, обычно используются команды RISC-типа, поэтому архитектуру VLIW иногда называют постRISC-архитектурой. Максимальное число полей в сверхдлинной команде равно числу вычислительных устройств и обычно колеблется в диапазоне от 3 до 20. Все вычислительные устройства имеют доступ к данным, хранящимся в едином многопортовом регистровом файле. Отсутствие сложных аппаратных механизмов, характерных для суперскалярных процессоров (предсказание переходов, внеочередное исполнение и т. д.), дает значительный выигрыш в быстродействии и возможность более эффективно использовать площадь кристалла. Подавляющее большинство цифровых сигнальных процессоров и мультимедийных процессоров с производительностью более 1 млрд операций/с базируется на VLIW-архитектуре. Серьезная проблема VLIW - усложнение регистрового файла и связей этого файла с вычислительными устройствами.

Вычислительные системы с явным параллелизмом команд (EPIC)

Дальнейшим развитием идеи VLIW стала новая архитектура IA-64 — совместная разработка фирм Intel и Hewlett-Packard (IA - это аббревиатура от Intel Architecture). В IA-64 реализован новый подход, известный как *вычисления с явным параллелизмом команд* (EPIC, Explicitly Parallel Instruction Computing) и являющийся усовершенствованным вариантом технологии VLIW. Первым представителем данной стратегии стал микропроцессор Itanium компании Intel. Корпорация Hewlett-Packard также реализует данный подход в своих разработках.

В архитектуре IA-64 предполагается наличие в процессоре 128 64-разрядных регистров общего назначения (РОН) и 128 80-разрядных регистров с плавающей запятой. Кроме того, процессор IA-64 содержит 64 однобитовых регистра предикатов.

Формат команд в архитектуре IA-64 показан на рис. 13.27.

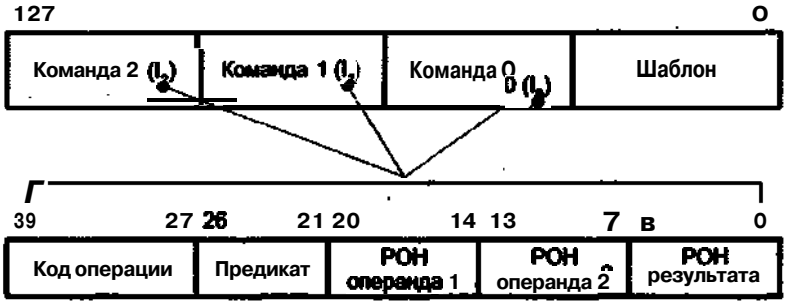


Рис. 13.27. Формат сверхдлинной команды в архитектуре IA-64

Команды упаковываются (группируются) компилятором в сверхдлинную команду — *связку* (bundle) длиною в 128 разрядов. Связка содержит три команды

и шаблон, в котором указываются зависимости между командами (можно ли с командой I_0 запустить параллельно I1 или же I1 должна выполниться только после Io), а также между другими связками (можно ли с командой I_2 из связки S_o запустить параллельно команду I_3 из связки Sl).

Перечислим все варианты составления связки из трех команд:

- $I_0 \| I_1 \| 1_2$ — все команды исполняются параллельно;
- Io & *I1* $\| I_2$ — сначала I_0, затем исполняются параллельно I1 и I_2;
- $I_0 \| I_1$ & I_2 - параллельно обрабатываются I_0 и I1 после них - I_2;
- I_0 & I1 & I_2 — команды исполняются в последовательности I_0, I_1, I_2.

Одна связка, состоящая из трех команд, соответствует набору из трех функциональных блоков процессора. Процессоры IA-64 могут содержать разное количество таких блоков, оставаясь при этом совместимыми по коду. Благодаря тому что в шаблоне указана зависимость и между связками, процессору с *N* одинаковыми блоками из трех ФБ будет соответствовать сверхдлинная команда из *N x* 3 команд (Nсвязок). Тем самым обеспечивается масштабируемость IA-64.

Поле каждой из трех команд в связке, в свою очередь, состоит из пяти полей:

- 13-разрядного поля кода операции;
- 6-разрядного поля предикатов, хранящего номер одного из 64 регистров предиката;
- 7-разрядного поля первого операнда (первого источника), где указывается номер регистра общего назначения или регистра с плавающей запятой, в котором содержится первый операнд;
- 7-разрядного поля второго операнда (второго источника), где указывается номер регистра общего назначения или регистра с плавающей запятой, в котором содержится второй операнд;
- 7-разрядного поля результата (приемника), где указывается номер регистра общего назначения или регистра с плавающей запятой, куда должен быть занесен результат выполнения команды.

Следует пояснить роль подполя предикатов. *Предикация* - это способ обработки условных ветвлений. Суть в том, что еще компилятор указывает, что обе ветви выполняются на процессоре параллельно, ведь EPIC-процессоры должны иметь много функциональных блоков.

Если в исходной программе встречается условное ветвление (по статистике — через каждые шесть команд), то команды из разных ветвей помечаются разными регистрами предиката (команды имеют для этого соответствующие поля), далее они выполняются совместно, но их результаты не записываются, пока значения регистров предиката (РП) не определены. Когда, наконец, вычисляется условие ветвления, РП, соответствующий «правильной» ветви, устанавливается в 1, а другой - в 0. Перед записью результатов процессор проверяет поле предиката и записывает результаты только тех команд, поле предиката которых указывает на РП с единичным значением.

Предикаты формируются как результат сравнения значений, хранящихся в двух регистрах. Результат сравнения («Истина» или «Ложь») заносится в один из РП,

но одновременно с этим во второй РП записывается инверсное значение полученного результата. Такой механизм позволяет процессору более эффективно выполнять конструкции типа IF-THEN-ELSE.

Логика выдачи команд на исполнение сложнее, чем в традиционных процессорах типа VLIW, но намного проще, чем у суперскалярных процессоров с неупорядоченной выдачей. По мнению специалистов Intel и HP, концепция EPIC, сохраняя все достоинства архитектурной организации VLIW, свободна от большинства ее недостатков. Особенностями архитектуры EPIC являются:

- большое количество регистров;
- масштабируемость архитектуры до большого количества функциональных блоков. Это свойство представители компаний Intel и HP называют *наследственно масштабируемой системой команд (Inherently Scaleable Instruction Set)*;
- явный параллелизм в машинном коде. Поиск зависимостей между командами осуществляет не процессор, а компилятор;
- предикация — команды из разных ветвей условного предложения снабжаются полями предикатов (полями условий) и запускаются параллельно;
- предварительная загрузка — данные из медленной основной памяти загружаются заранее.

Общий итог обзора технологии VLIW можно сформулировать следующим образом.

Преимущества. Использование компилятора позволяет устранить зависимости между командами до того, как они будут реально выполняться, в отличие от суперскалярных процессоров, где такие зависимости приходится обнаруживать и устранять "на лету". Отсутствие зависимостей между командами в коде, сформированном компилятором, ведет к упрощению аппаратных средств процессора и за счет этого к существенному подъему его быстродействия. Наличие множества функциональных блоков дает возможность выполнять несколько команд параллельно.

Недостатки. Требуется новое поколение компиляторов, способных проанализировать программу, найти в ней независимые команды, связать такие команды в строки длиной от 256 до 1024 бит, обеспечить их параллельное выполнение. Компилятор должен учитывать конкретные детали аппаратных средств. При определенных ситуациях программа оказывается недостаточно гибкой.

Основные сферы применения. VLIW-процессоры пока еще распространены относительно мало. Основными сферами применения технологии VLIW-являются цифровые сигнальные процессоры и вычислительные системы, ориентированные на архитектуру IA-64. Наиболее известной была VLIW-система фирмы Multiflow Computer, Inc. (Уже не существующей.) В России VLIW-концепция была реализована в суперкомпьютере Эльбрус 3-1 и получила дальнейшее развитие в его последователе - Эльбрус-2000 (Е2к). К VLIW можно причислить семейство сигнальных процессоров TMS320C6x фирмы Texas Instruments. С 1986 года ведутся исследования VLIW-архитектуры в IBM. В начале 2000 года фирма Transmeta заявила процессор Crusoe, представляющий собой программно-аппаратный комплекс. В нем команды микропроцессоров серии x86 транслируются в слова VLIW длиной 64 или 128 бит. Оттранслированные команды хранятся в кэш-памяти, а трансляция при многократном их использовании производится только один раз. Ядро процессора исполняет элементы кода в строгой последовательности.

Контрольные вопросы

1. Какой уровень параллелизма в обработке информации обеспечивают вычислительные системы класса SIMD?
2. На какие структуры данных ориентированы средства векторной обработки?
3. Благодаря чему многомерные массивы при обработке можно рассматривать в качестве одномерных векторов?
4. Поясните различие между конвейерными и векторно-конвейерными вычислительными системами.
5. Поясните назначение регистров векторного процессора: регистра длины вектора, регистра максимальной длины вектора, регистра вектора индексов и регистра маски.
6. Для чего используются операции упаковки/распаковки вектора?
7. Оцените выигрыш в быстродействии векторного процессора за счет сцепления векторов.
8. В чем заключается принципиальное различие между векторными и матричными вычислительными системами?
9. Какими средствами обеспечивается подготовка программ для матричных вычислительных систем и их загрузка?
10. По какому принципу в матричной ВС команды программы распределяются между центральным процессором и массивом процессоров?
11. Каким образом в матричной ВС реализуются предложения типа IF-THEN-ELSE?
12. Как идентифицируются отдельные процессорные элементы в массиве процессоров матричной ВС?
13. Какие схемы глобального маскирования применяются в матричных ВС и в каких случаях каждая из них является предпочтительной?
14. Могут ли участвовать в вычислениях замаскированные (пассивные) процессорные элементы матричной ВС и в каком виде это участие проявляется?
15. Поясните различие между ассоциативной памятью и ассоциативным процессором.
16. В чем выражается аналогия между матричными и ассоциативными ВС?
17. Какую особенность систолической ВС отражает ее название?
18. Объясните достоинства и недостатки систолических массивов типа ULA, BLA, TLA.
19. Сформулируйте правила объединения простых команд в командное слово сверхбольшой длины.
20. Чем ограничивается количество объединяемых команд в технологии EPIC?
21. Поясните назначение системы предикации и ее реализацию в архитектуре IA-64.

Глава 14

Вычислительные системы класса MIMD

Технология SIMD исторически стала осваиваться раньше, что и предопределило широкое распространение SIMD-систем. В настоящее время тем не менее наметился устойчивый интерес к архитектурам класса MIMD. MIMD-системы обладают большей гибкостью, в частности могут работать и как высокопроизводительные однопользовательские системы, и как многопрограммные ВС, выполняющие множество задач параллельно. Кроме того, архитектура MIMD позволяет наиболее эффективно распорядиться всеми преимуществами современной микропроцессорной технологии,

В MIMD-системе каждый процессорный элемент (ПЭ) выполняет свою программу достаточно независимо от других ПЭ. В то же время ПЭ должны как-то взаимодействовать друг с другом. Различие в способе такого взаимодействия определяет условное деление MIMD-систем на ВС с общей памятью и системы с распределенной памятью. В *системах с общей памятью,* которые характеризуют как *сильно связанные* (tightly coupled), имеется общая память данных и команд, доступная всем процессорным элементам с помощью общей шины или сети соединений. К этому типу, в частности, относятся *симметричные мультипроцессоры (SMP, Symmetric Multiprocessor) и системы с неоднородным доступом к памяти* (NUMA, Non-Uniform Memory Access).

В *системах с распределенной памятью* или *слабо связанных* (loosely coupled) многопроцессорных системах вся память распределена между процессорными элементами, и каждый ёлок памяти доступен только «своему» процессору. Сеть соединений связывает процессорные элементы друг с другом. Представителями этой *группы могут служ*ить *системы с массовым параллелизмом* (MPP, Massively Parallel Processing) и *кластерные вычислительные системы.*

Базовой моделью вычислений на MIMD-системе является совокупность независимых процессов, эпизодически обращающихся к совместно используемым данным. Существует множество вариантов этой модели. На одном конце спектра - распределенные вычисления, в рамках которых программа делится на довольно большое число параллельных задач, состоящих из множества подпрограмм.

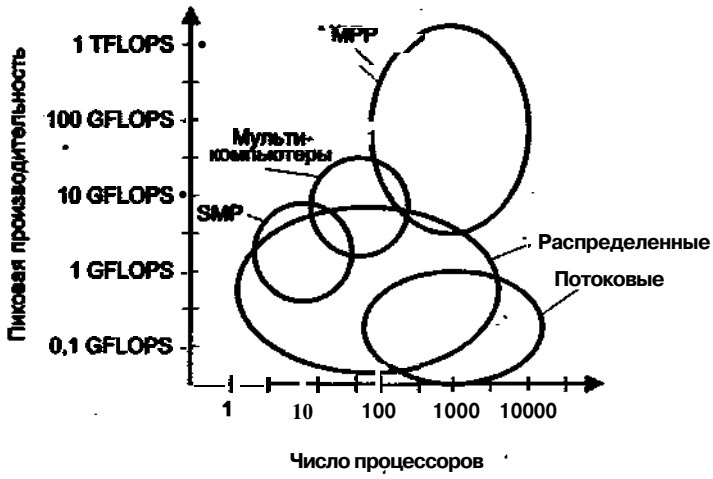


Рис. 1 4 . 1 . Производительность MIMD-систем как функция их типа и числа процессоров

На другом конце — модель потоковых вычислений, где каждая операция в программе может рассматриваться как отдельный процесс. Такая операция ожидает поступления входных данных (операндов), которые должны быть переданы ей другими процессами. По их получении операция выполняется, и результирующее значение передается тем процессам, которые в нем нуждаются. Примерные значения пиковой производительности для различных типов систем класса MIMD показаны на рис. 14.1.

Симметричные мультипроцессорные системы

До сравнительно недавнего времени практически все однопользовательские персональные ВМ и рабочие станции содержали по одному микропроцессору общего назначения. По мере возрастания требований к производительности и снижения стоимости микропроцессоров поставщики вычислительных средств как альтернативу *о*д н*опроцессорн*ы м *ВМ ст*а л и п*редла*г а т*ь симметричные мультипроцессорные вычислительные системы, так* н а з ы в а е*мые SMP-системы (SMP, Symmet*r ic *Mu*lti*processor)*. Это понятие относится как к архитектуре ВС, так и к поведению операционной системы, отражающему данную архитектурную организацию. SMP можно определить как вычислительную систему, обладающую следующими характеристиками:

- Имеются два или более процессоров сопоставимой производительности.
- Процессоры совместно используют основную память и работают в едином виртуальном и физическом адресном пространстве.
- Все процессоры связаны между собой посредством шины или по иной схеме, так что время доступа к памяти любого из них одинаково.
- Все процессоры разделяют доступ к устройствам ввода/вывода либо через одни и те же каналы, либо через разные каналы, обеспечивающие доступ к одному и тому же внешнему устройству.

и Все процессоры способны выполнять одинаковые функции (этим объясняется термин «симметричные»),

- Любой из процессоров может обслуживать внешние прерывания.

-Вычислительная система управляется интегрированной операционной системой, которая организует и координирует взаимодействие между процессорами и программами на уровне заданий, задач, файлов и элементов данных.

Обратим внимание па последний пункт, который подчеркивает одно из отличий по отношению к слабо связанным мультипроцессорным системам, таким как кластеры, где в качестве физической единицы обмена информацией обычно выступает сообщение или полный файл. В SMP допустимо взаимодействие на уровне отдельного элемента данных, благодаря чему может быть достигнута высокая степень связности между процессами.

Хотя технически SMP-системы симметричны, в их работе присутствует небольшой фактор перекоса, который вносит программное обеспечение. На время загрузки системы один из процессоров получает статус ведущего (master). Это не означает, что позже, во время работы какие-то процессоры будут ведомыми - все они в SMP-системе равноправны. Термин «ведущий» вводится только затем, чтобы указать, какой из процессоров по умолчанию будет руководить первоначальной загрузкой ВС.

Операционная система планирует процессы или нити процессов (threads) сразу по всем процессорам, скрывая при этом от пользователя многопроцессорный характер SMP-архитектуры.

По сравнению с однопроцессорными схемами SMP-системы имеют преимущество по следующим показателям [36]:

- Производительность. Если подлежащая решению задача поддается разбиению на несколько частей так, что отдельные части могут выполняться параллельно, то множество процессоров дает выигрыш в производительности относительно одиночного процессора того же типа (рис. 14.2),

- Готовность. В симметричном мультипроцессоре отказ одного из компонентов не ведет к отказу системы, поскольку любой из процессоров в состоянии выполнять те же функции, что и другие.

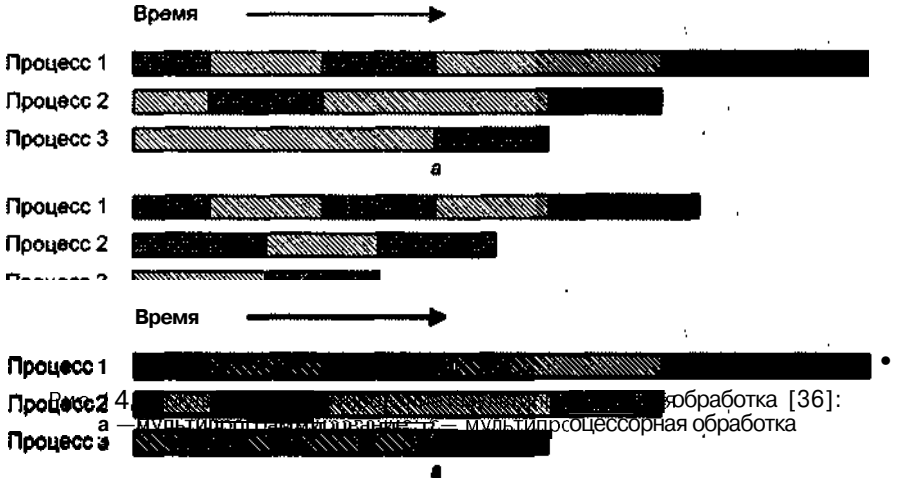


Рис. 14.2. обработка [36]:
а — мультипроцессорная обработка

- **Расширяемость.** Производительность системы может быть увеличена добавлением дополнительных процессоров.
- Масштабируемость. Варьируя число процессоров в системе, можно создать системы различной производительности и стоимости.

Необходимо отметить, что перечисленное — это только потенциальные преимущества, реализация которых невозможна, если в операционной системе отсутствуют средства; для поддержки параллелизма.

Архитектура SMP-системы

На рис. 14.3 в самом общем виде показана архитектура симметричной мультипроцессорной ВС.

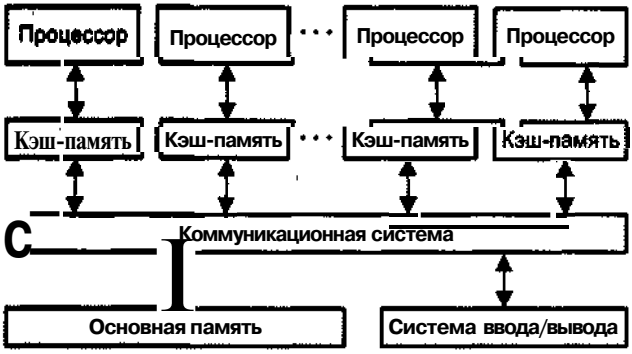


Рис. 14.3. Организация симметричной мультипроцессорной системы

Типовая SMP-система содержит от двух до 32 идентичных процессоров, в качестве которых обычно выступают недорогие RISC-процессоры, такие, например, как DEC Alpha, Sun SPARC, MIPS или HP PA-RISC. В последнее время наметилась тенденция оснащения SMP-систем также и CISC-процессорами, в частности Pentium.

Каждый процессор снабжен локальной кэш-памятью, состоящей из кэш-памяти первого (L1) и второго (L2) уровней. Согласованность содержимого кэш-памяти всех процессоров обеспечивается аппаратными средствами. В некоторых SMP-системах проблема когерентности снимается за счет совместно используемой кэш-памяти (рис. 14.4). К сожалению., этот прием технически и экономически оправдан лишь, если число процессоров не превышает четырех. Применение общей кэш-памяти сопровождается повышением стоимости и снижением быстродействия кэш-памяти.

Все процессоры ВС имеют равноправный доступ к разделяемым основной памяти и устройствам ввода/вывода. Такая возможность обеспечивается коммуникационной системой. Обычно процессоры взаимодействуют между собой через основную память (сообщения и информация о состоянии оставляются в области общих данных). В некоторых SMP-системах предусматривается также прямой обмен сигналами между процессорами.

Память системы обычно строится по модульному принципу и организована так, что допускается одновременное обращение к разным ее модулям (банкам). В неко-

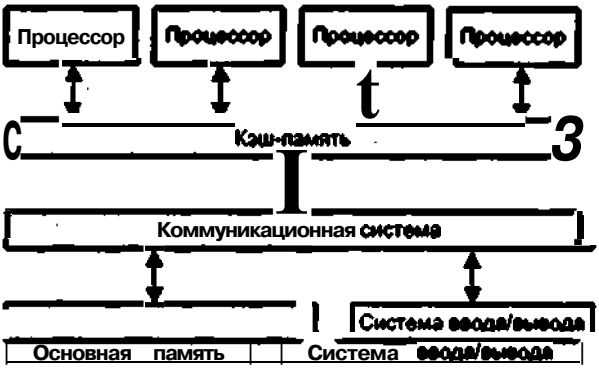


Рис. 14.4. SMP-система с совместно используемой кэш-памятью

торых конфигурациях в дополнение к совместно используемым ресурсам каждый процессор обладает также собственными локальной основной памятью и каналами ввода/вывода.

Важным аспектом архитектуры симметричных мультипроцессоров является способ взаимодействия процессоров с общими ресурсами (памятью и системой ввода/вывода). С этих позиций можно выделить следующие виды архитектуры SMP-систем:

- с общей шиной и временным разделением;
- с коммутатором типа «кроссбар»;
- с многопортовой памятью;
- *с* централизованным устройством управления.

Архитектура с общей шиной

Структура и интерфейсы общей шины в основном такие же, как и в однопроцессорной ВС, где шина служит для внутренних соединений (рис. 14.5).

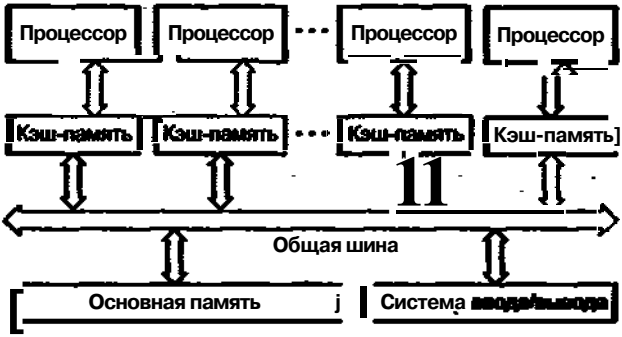


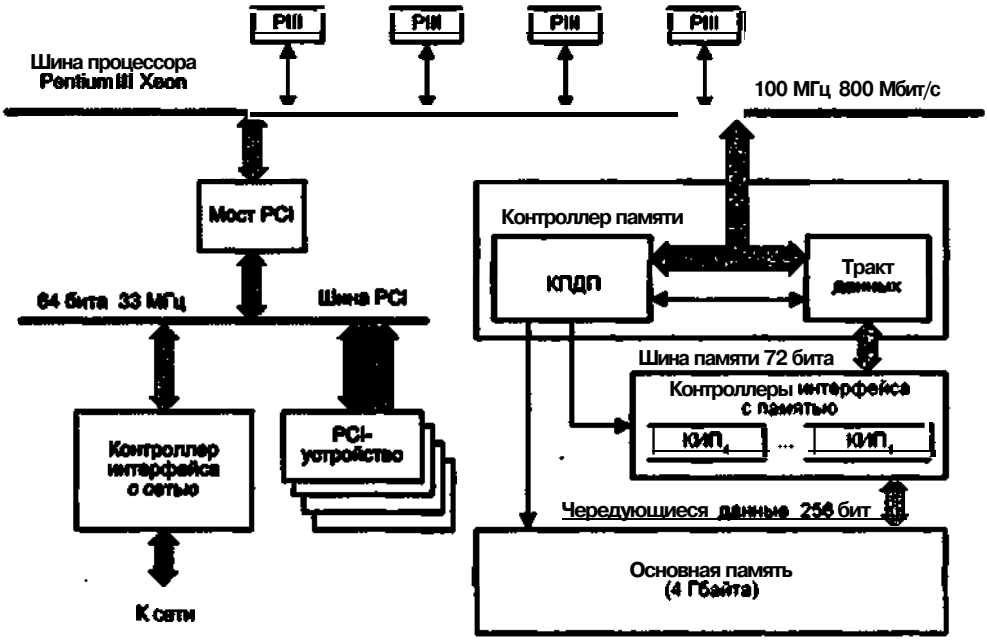
Рис. 14.5. Структура SMP-системы с общей шиной

Достоинства и недостатки систем коммуникации на базе общей шины с разделением времени достаточно подробно обсуждались ранее. Применительно к SMP-системам отметим, что физический интерфейс, а также логика адресации, арбитража и разделения времени остаются теми же, что и в однопроцессорных системах.

Общая шина позволяет легко расширять систему путем подключения к себе большего числа процессоров. Кроме того, напомним, что шина — это, по существу, пассивная среда, и отказ одного из подключенных к ней устройств не влечет отказа всей совокупности.

В то же время SMP-системам на базе общей шины свойственен и основной недостаток шинной организации — невысокая производительность: скорость системы ограничена временем цикла шины. По этой причине каждый процессор снабжен кэш-памятью, что существенно уменьшает число обращений к шине. Наличие множества кэшей порождает проблему их когерентности, и это одна из основных причин, по которой системы на базе общей шины обычно содержат не слишком много процессоров. Так, в системах Compaq AlphaServer GS140 и 8400 используется неболее 14 процессоров Alpha 21264. SMP-система HPN9000 в максимальном варианте состоит из 8 процессоров PA-8500, а система SMP Thin Nodes для RS/6000 фирмы IBM может включать в себя от двух до четырех процессоров PowerPC 604.

Архитектура с общей шиной широко распространена в SMP-системах, построенных на микропроцессорах x86. В эту группу входят: DELL Power Edge, IBM Netfinity, HP NetServer. На рис. 14.6 показана структура симметричной мультипроцессорной вычислительной системы на базе микропроцессоров Pentium III.



Рис, 14.6. Структура SMP-системы на базе микропроцессоров Pentium III

Архитектура с коммутатором типа «кроссбар»

Архитектура с коммутатором типа «кроссбар»- (рис. 14.7) ориентирована на модульное построение общей памяти и призвана разрешить проблему ограниченной цропускной способности систем с общей шиной.

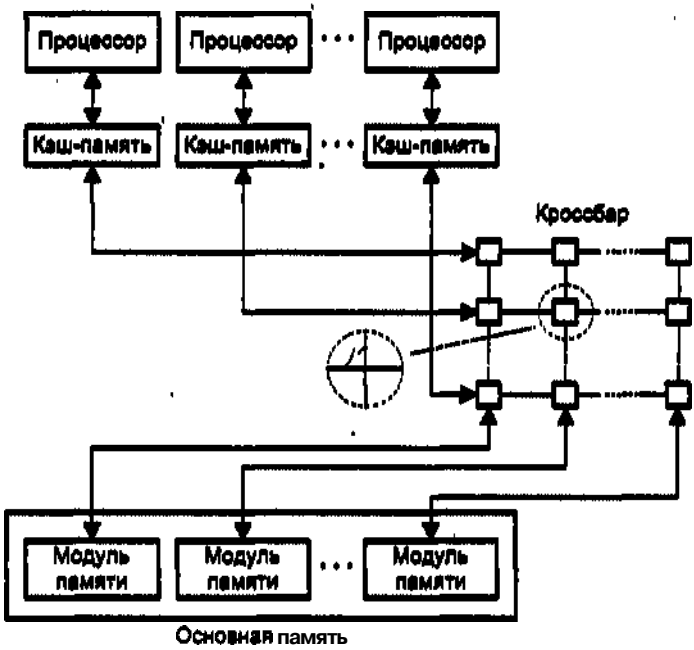


Рис. 14.7. Структура SMP-системы с коммутатором типа «кроссбар»

Коммутатор обеспечивает множественность путей между процессорами и банками памяти, причем топология связей может быть как двумерной, так и трехмерной. Результатом становится более высокая полоса пропускания, что позволяет строить SMP-системы, содержащие больше процессоров, чем в случае общей шины. Типичное число процессоров в SMP-системах на базе матричного коммутатора составляет 32 или 64. Отметим, что выигрыш в производительности достигается, лишь когда разные процессоры обращаются к разным банкам памяти.

По логике кроссбара строится и взаимодействие процессоров с устройствами ввода/вывода.

В качестве примера ВС с рассмотренной архитектурой можно привести систему Enterprise 10000, состоящую из 64 процессоров, связанных с памятью посредством матричного коммутатора Gigaplane-XB фирмы Sun Microsystems (кроссбар 16 x 16), В IBM RS/6000 Enterprise Server Model S70 коммутатор типа «кроссбар» обеспечивает работу 12 процессоров RS64. В SMP-системах ProLiant 8000 и 8500 фирмы Compaq для объединения с памятью и между собой восьми процессоров Pentium III Хеоп применена комбинация нескольких.шин и кроссбара.

Концепция матричного коммутатора (кроссбара) не ограничивается симметричными мультипроцессорами. Аналогичная структура связей применяется для объединения узлов в ВС типа CC-NUMA и кластерных вычислительных системах.

Архитектура с многопортовой памятью

Многопортовая организация запоминающего устройства обеспечивает любому процессору и модулю ввода/вывода прямой и непосредственный доступ к банкам основной памяти (ОП). Такой подход сложнее, чем при использовании шины,

поскольку требует придания ЗУ основной памяти дополнительной, достаточно сложной логики. Тем не менее это позволяет поднять производительность, так как каждый процессор имеет выделенный тракт к каждому модулю ОП. Другое преимущество многопортовой организации — возможность назначить отдельные модули памяти в качестве локальной памяти отдельного процессора. Эта особенность позволяет улучшить защиту данных от несанкционированного доступа со стороны других процессоров.

Архитектура с централизованным устройством управления

Централизованное устройство управления (ЦУУ) сводит вместе отдельные потоки данных между независимыми модулями: процессором, памятью, устройствами ввода/вывода. ЦУУ может буферйзировать запросы, выполнять синхронизацию и арбитраж. Оно способно передавать между процессорами информацию о состоянии и управляющие сообщения, а также предупреждать об изменении информации в кэшах. Недостаток такой организации заключается в сложности устройства управления, что становится потенциальным узким местом в плане производительности. В настоящее время подобная архитектура встречается достаточно редко, но она широко использовалась при создании вычислительных систем на базе машин семейства IBM 370.

Кластерные вычислительные системы

Одно из самых современных направлений в области создания вычислительных систем — это *кластеризация*. По производительности и коэффициенту готовности кластеризация представляет собой альтернативу симметричным мультипроцессорным системам. Понятие *кластер* определим как группу взаимно соединенных вычислительных систем (узлов), работающих совместно, составляя единый вычислительный ресурс и создавая иллюзию наличия единственной ВМ. В качестве узла кластера может выступать как однопроцессорная ВМ, так и ВС типа SMP или MPP. Важно лишь то, что каждый узел в состоянии функционировать самостоятельно и отдельно от кластера. В плане архитектуры суть кластерных вычислений сводится к объединению нескольких узлов высокоскоростной сетью. Для описания такого подхода, помимо термина «кластерные вычисления», достаточно часто применяют такие названия, как: *кластер рабочих станций* (workstation cluster), *гипервычисления* (hypercomputing), *параллельные вычисления на базе сети* (network-based concurrent computing), *ультравычисления* (ultracomputing).

Изначально перед кластерами ставились две задачи: достичь большой вычислительной мощности и обеспечить повышенную надежность ВС. Пионером в области кластерных архитектур считается корпорация DEC, создавшая первый коммерческий кластер в начале 80-х годов прошлого века.

В качестве узлов кластеров могут использоваться как одинаковые ВС (гомогенные кластеры), так и разные (гетерогенные кластеры). По своей архитектуре кластерная ВС является слабо связанной системой.

В работе [65] перечисляются четыре преимущества, достигаемые с помощью кластеризации:

- **Абсолютная масштабируемость.** Возможно создание больших кластеров, превосходящих по вычислительной мощности даже самые производительные одиночные ВМ. Кластер в состоянии содержать десятки узлов, каждый из которых представляет собой мультипроцессор.
- **Наращиваемая масштабируемость.** Кластер строится так, что его можно наращивать, добавляя новые узлы небольшими порциями. Таким образом, пользователь может начать с умеренной системы, расширяя ее по мере необходимости.
- **Высокий коэффициент готовности.** Поскольку каждый узел кластера — самостоятельная ВМ или ВС, отказ одного из узлов не приводит к потере работоспособности кластера. Во многих системах отказоустойчивость автоматически поддерживается программным обеспечением.
- **Превосходное соотношение цена/производительность.** Кластер любой производительности можно создать, соединяя стандартные «строительные блоки»-, при этом его стоимость будет ниже, чем у одиночной ВМ с эквивалентной вычислительной мощностью.

На уровне аппаратного обеспечения кластер — это просто совокупность независимых вычислительных систем, объединенных сетью. При соединении машин в кластер почти всегда поддерживаются прямые межмашинные связи. Решения могут быть простыми, основывающимися на аппаратуре Ethernet, или сложными с высокоскоростными сетями с пропускной способностью в сотни мегабайтов в секунду. К последней категории относятся RS/6000 SP компании IBM, системы фирмы Digital на основе Memory Channel, ServerNet корпорации Compaq.

Узлы кластера контролируют работоспособность друг друга и обмениваются специфической, характерной для кластера информацией. Контроль работоспособности осуществляется с помощью специального сигнала, часто называемого *heartbeat,* что можно перевести как «сердцебиение». Этот сигнал передается узлами кластера друг другу, чтобы подтвердить их нормальное функционирование.

Неотъемлемая часть кластера — специализированное программное обеспечение (ПО), на которое возлагается задача обеспечения бесперебойной работы при отказе одного или нескольких узлов. Такое ПО производит перераспределение вычислительной нагрузки при отказе одного или нескольких узлов кластера, а также восстановление вычислений при сбое в узле. Кроме того, при наличии в кластере совместно используемых дисков кластерное ПО поддерживает единую файловую систему.

Классификация архитектур кластерных систем

В литературе приводятся различные способы классификации кластеров. Так, в простейшем варианте ориентируются на то, являются ли диски в кластере разделяемыми всеми узлами. На рис. 14.8, *а* показан кластер из двух узлов, совместная работа которых координируется за счет высокоскоростной линии, по которой происходит обмен сообщениями. Такой линией может быть локальная сеть, используемая также и не входящими в кластер компьютерами, либо выделенная линия.

В последнем случае один или несколько узлов кластера будут иметь выход на локальную или глобальную сеть, благодаря чему обеспечивается связь между серверным кластером и удаленными клиентскими системами.

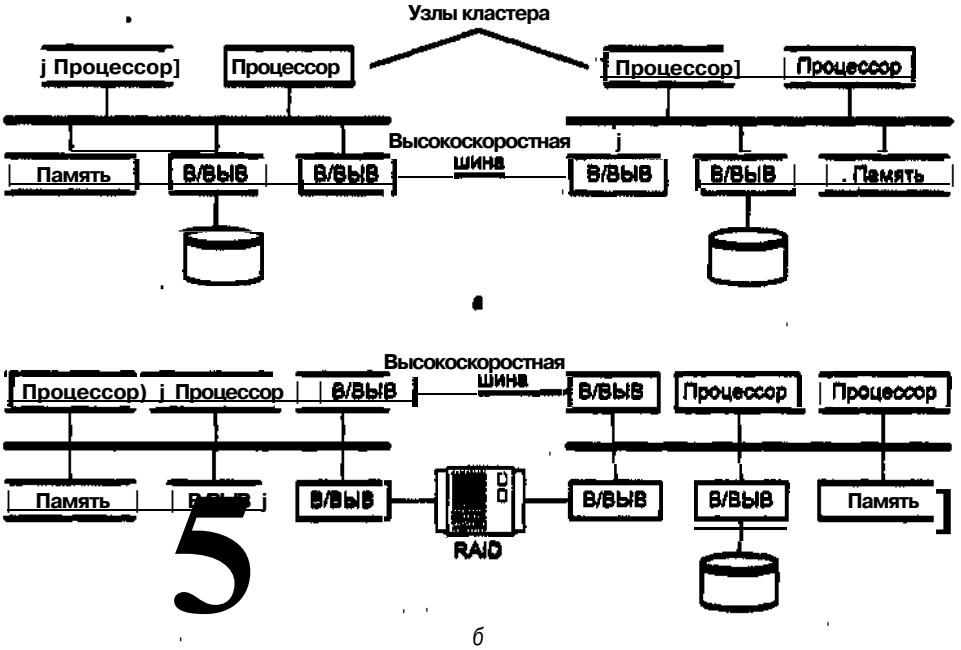


Рис. **14.8.** Конфигурации кластеров: а - без совместно используемых дисков; *б* — с совместно используемыми дисками

Более ясную картину дает группировка кластеров на основе сходства их функциональных особенностей. Такая классификация приведена в табл. 14.1.

Таблица 14.1. Методы кластеризации

Метод кластеризации	Описание
Пассивное резервирование	Вторичный сервер при отказе первичного берет управление на себя
Резервирование с активным вторичным сервером	Вторичный сервер, как и первичный, используется при решении задач
Самостоятельные серверы	Самостоятельные серверы имеют собственные диски, а данные постоянно копируются с первичного сервера на вторичный
Серверы с подключением ко всем дискам	Серверы подключены к одним и тем же дискам, но каждый сервер владеет своей их частью. Если один из серверов отказывает, то управление его дисками берет на себя другой сервер
Серверы с совместно используемыми дисками	Множество серверов работают в режиме коллективного доступа к дискам

Кластеризация с резервированием - наиболее старый и универсальный метод. Один из серверов берет на себя всю вычислительную нагрузку, в то время как второй остается неактивным, но готовым перенять вычисления при отказе основного сервера. Активный или первичный сервер периодически посылает резервному тактирующее сообщение. При отсутствии тактирующих сообщений (это рассматривается как отказ первичного сервера) вторичный сервер берет управление на себя. Такой подход повышает коэффициент готовности, но не улучшает производительности. Более того, если единственный вид взаимодействия между узлами - обмен сообщениями, и если оба сервера кластера не используют диски коллективно, то резервный сервер не имеет доступа к базам данных, управляемым первичным сервером.

Пассивное резервирование для кластеров не характерно. Термин «кластер» относят к множеству взаимосвязанных узлов, активно участвующих в вычислительном процессе и совместно создающих иллюзию одной мощной вычислительной машины. К такой конфигурации обычно применяют понятие системы *с активным вторичным сервером,* и здесь выделяют три метода кластеризации: самостоятельные серверы, серверы без совместного использования дисков и серверы с совместным использованием дисков.

При первом подходе каждый узел кластера рассматривается как самостоятельный сервер с собственными дисками, причем ни один из дисков в системе не является общим (см. рис. 14.8, я). Схема обеспечивает высокую производительность и высокий коэффициент готовности, однако требует специального программного обеспечения для планирования распределения клиентских запросов по серверам так, чтобы добиться сбалансированной и эффективной нагрузки на каждый из них. Необходимо также создать условия, чтобы при отказе одного из узлов в процессе выполнения какого-либо приложения другой узел мог перехватить и завершить оставшееся без управления приложение. Для этого данные в кластере должны постоянно копироваться, чтобы каждый сервер имел доступ ко всем наиболее свежим данным в системе. Из-за этих издержек высокий коэффициент готовности достигается лишь за счет потери производительности.

Для сокращения коммуникационных издержек большинство кластеров в настоящее время формируют из серверов, подключенных к общим дискам, обычно представленных дисковым массивом RAID (рис. 14.8,*б).*

Один из вариантов такого подхода предполагает, что совместный доступ к дискам не применяется. Общие диски разбиваются на разделы, и каждому узлу кластера выделяется свой раздел. Если один из узлов отказывает, кластер может быть реконфигурирован так, что права доступа к его части общего диска передаются другому узлу.

Во втором варианте множество серверов разделяют во времени доступ к общим дискам, так что любой узел имеет возможность обратиться к любому разделу каждого общего диска. Эта организация требует наличия каких-либо средств блокировки, гарантирующих, что в любой момент времени доступ к данным будет иметь только один из серверов.

Вычислительные машины (системы) в кластере взаимодействуют в соответствии с одним их двух транспортных протоколов. Первый из них, протокол TCP (Transmission Control Protocol), оперирует потоками байтов, гарантируя надеж-

ность доставки сообщения. Второй - UDP (User Datagram Protocol) пытается посылать пакеты данных без гарантии их доставки, В последнее время применяют специальные протоколы, которые работают намного лучше. Так, возглавляемый компанией Intel консорциум (Microsoft, Compaq и др.) предложил новый протокол для внутрикластерных коммуникаций, который называется Virtual Interface Architecture (VIA) и претендует на роль стандарта.

При обмене информацией используются два программных метода: *передачи сообщений и распределенной совместно используемой памяти. Перв*ый оп*ирается* на явную передачу информационных сообщений между узлами кластера. В альтернативном варианте также происходит пересылка сообщений, но движение данных между узлами кластера скрыто от программиста.

Кластеры обеспечивают высокий уровень доступности - в них отсутствуют единая операционная система и совместно используемая память, то есть нет проблемы когерентности кэшей. Кроме того, специальное программное обеспечение в каждом узле постоянно контролирует работоспособность всех остальных узлов. Этот контроль основан на периодической рассылке каждым узлом сигнала «Пока жив» (keepalive). Если сигнал от некоторого узла не поступает, то такой узел считается вышедшим из строя; ему не дается возможность выполнять ввод/вывод, его диски и другие ресурсы (включая сетевые адреса) переназначаются другим узлам, а выполнявшиеся им программы перезапускаются в других узлах.

Кластеры хорошо масштабируются в плане производительности при добавлении узлов. В кластере может выполняться несколько отдельных приложений, но для масштабирования отдельного приложения требуется, чтобы его части согласовывали свою работу путем обмена сообщениями. Нельзя, однако, не учитывать, что взаимодействия между узлами кластера занимают гораздо больше времени, чем в традиционных ВС.

Возможность практически неограниченного наращивания числа узлов и отсутствие единой операционной системы делают кластерные архитектуры исключительно успешно масштабируемыми, и даже системы с сотнями и тысячами узлов показывают себя на практике с положительной стороны.

Топологии кластеров

При создании кластеров с большим количеством узлов могут применяться самые разнообразные топологии (см. главу 12). В данном разделе остановимся на тех, которые характерны для наиболее распространенных «малых»- кластеров, состоящих из 2-4 узлов.

Топология кластерных пар

Топология кластерных пар находит применение при организации двух- или четырехузловых кластеров (рис. 14.9).

Узлы группируются попарно. Дисковые массивы присоединяются к обоим узлам пары, причем каждый узел имеет доступ ко всем дисковым массивам своей пары. Один из узлов является резервным для другого.

Четырехузловая кластерная «пара»- представляет собой простое расширение двухузловой топологии. Обе кластерные пары с точки зрения администрирования и настройки рассматриваются как единое целое.

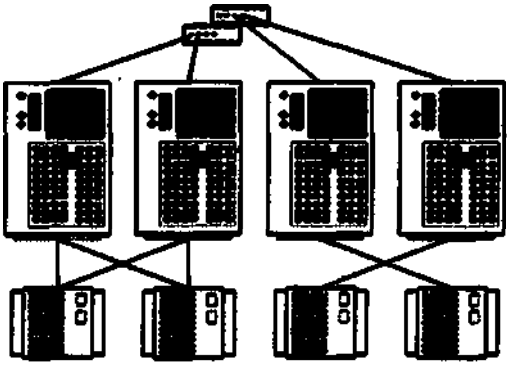


Рис. 14.9. Топология кластерных пар

Эта топология подходит для организации кластеров с высокой готовностью данных, но отказоустойчивость реализуется только в пределах пары, так как принадлежащие ей устройства хранения информации не имеют физического соединения с другой парой.

Пример: организация параллельной работы СУБД Informix XPS.

Топология N +1

Топология *N*+ 1 позволяет создавать кластеры из2, 3и4 узлов (рис. 14.10).

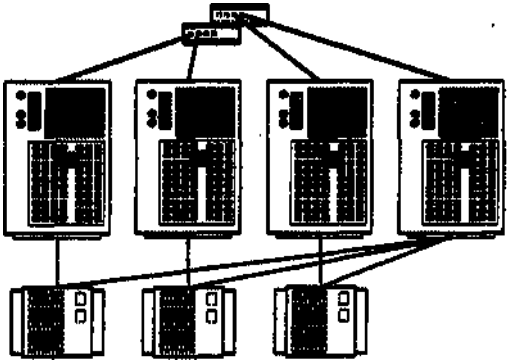


Рис. 14.10. Топология N +1

Каждый дисковый массив подключаются только к двум узлам кластера. Дисковые массивы организованы по схеме RAID 1. Один сервер имеет соединение со всеми дисковыми массивами и служит в качестве резервного для всех остальных (основных или активных) узлов. Резервный сервер может использоваться для поддержания высокой степени готовности в паре с любым из активных узлов.

Топология рекомендуется для организации кластеров высокой готовности. В тех конфигурациях, где имеется возможность выделить один узел для резервирования, эта топология способствует уменьшению нагрузки на активные узлы и гарантирует, что нагрузка вышедшего из строя узла будет воспроизведена на резервном узле без потери производительности. Отказоустойчивость обеспечивается между

любым из основных узлов и резервным узлом. В то же время топология не позволяет реализовать глобальную отказоустойчивость, поскольку основные узлы кластера и их системы хранения информации не связаны друг с другом.

Топология N *x* N

Аналогично топологии *N*+ 1, топология *Nx* N(рис. 14,11) рассчитана на создание кластеров из 2, 3 и 4 узлов, но в отличие от первой обладает большей гибкостью и масштабируемостью.

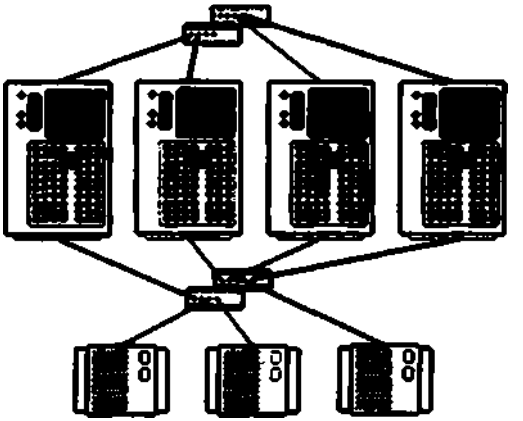


Рис. 14.11. Топология NxN

Только в этой топологии все узлы кластера имеют доступ ко всем дисковым массивам, которые, в свою очередь, строятся по схеме RAID 1 (с дублированием). Масштабируемость проявляется в простоте добавления к кластеру дополнительных узлов и дисковых массивов без изменения соединений в существующей системе.

Топология позволяет организовать каскадную систему отказоустойчивости, при которой обработка переносится с неисправного узла на резервный, а в случае его выхода из строя - на следующий резервный узел и т. д. Кластеры с топологией

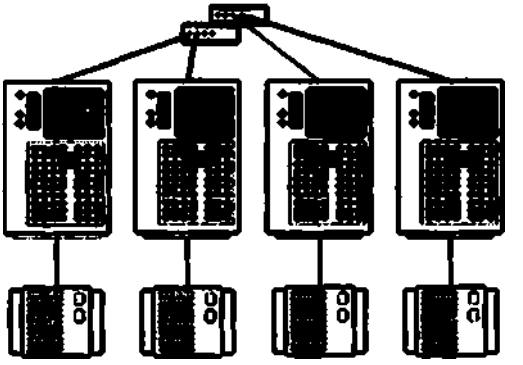


Рис. 14.12. Топология с полностью раздельным доступом

N x N обеспечивают поддержку приложения Oracle Parallel Server, требующего соединения всех узлов со всеми системами хранения информации. В целом топология характеризуется лучшей отказоустойчивостью и гибкостью по сравнению с другими решениями.

Топология с полностью раздельным доступом

В топологии с полностью раздельным доступом (рис. 14.12) каждый дисковый массив соединяется только с одним узлом кластера.

Топология рекомендуется только для тех приложений, для которых характерна архитектура полностью раздельного доступа, например для уже упоминавшейся СУБД Informix XPS.

Системы с массовой параллельной обработкой (MPP)

Основным признаком, по которому вычислительную систему относят к *архитектуре с массовой параллельной обработкой* (MPP, Massively Parallel Processing), служит количество процессоров *n*. Строгой границы не существует, но обычно при *n* >=128 считается, что это уже MPP, а при *n* <=32 - еще нет. Обобщенная структура MPP-системы показана на рис. 14.13.

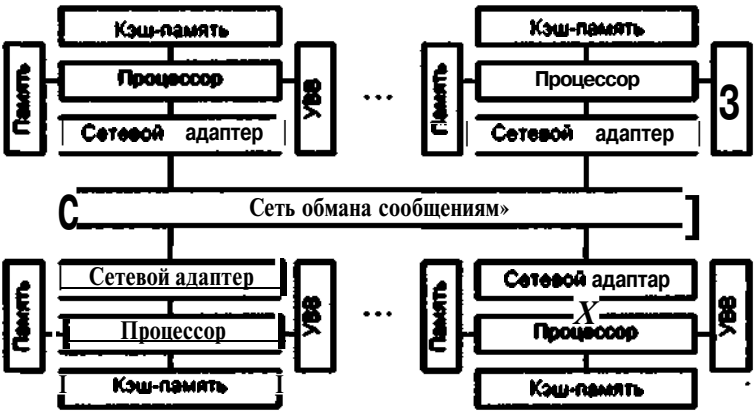


Рис. 14.13. Структура вычислительной системы с массовой параллельной обработкой

Главные особенности, по которым вычислительную систему причисляют к классу MPP, можно сформулировать следующим образом:

- стандартные микропроцессоры;
- физически распределенная память;
- сеть соединений с высокой пропускной способностью и малыми задержками;
- хорошая масштабируемость (до тысяч процессоров);
- асинхронная MIMD-система с пересылкой сообщений;
- программа представляет собой множество процессов, имеющих отдельные адресные пространства.

Основные причины появления систем с массовой параллельной обработкой - это, во-первых, необходимость построения ВС с гигантской производительностью и, во-вторых, стремление раздвинуть границы производства ВС в большом диапазоне, как производительности, так и стоимости. Для MPP-системы, в которой количество процессоров может меняться в широких пределах, всегда реально подобрать конфигурацию с заранее заданной вычислительной мощностью и финансовыми вложениями.

Если говорить о MPP как о представителе класса MIMD с распределенной памятью и отвлечься от организации ввода/вывода, то эта архитектура является естественным расширением кластерной на большое число узлов. Отсюда для MPP-систем характерны все преимущества и недостатки кластеров, причем в связи с повышенным числом процессорных узлов как плюсы, так и минусы становятся гораздо весомее.

Характерная черта MPP-систем — наличие единственного управляющего устройства (процессора), распределяющего задания между множеством подчиненных ему устройств, чаще всего одинаковых (взаимозаменяемых), принадлежащих одному или нескольким классам. Схема взаимодействия в общих чертах довольно проста:

- центральное управляющее устройство формирует очередь заданий, каждому из которых назначается некоторый уровень приоритета;
- по мере освобождения подчиненных устройств им передаются задания из очереди;
- подчиненные устройства оповещают центральный процессор о ходе выполнения задания, в частности о завершении выполнения или о потребности в дополнительных ресурсах;
- у центрального устройства имеются средства для контроля работы подчиненных процессоров, в том числе для обнаружения нештатных ситуаций, прерывания выполнения задания в случае появления более приоритетной задачи и т. п.

В некотором приближении имеет смысл считать, что на центральном процессоре выполняется ядро операционной системы (планировщик заданий), а на подчиненных ему — приложения. Подчиненность между процессорами может быть реализована как на аппаратном, так и на программном уровне.

Вовсе не обязательно, чтобы MPP-система имела распределенную оперативную память, когда каждый процессорный узел владеет собственной локальной памятью. Так, например, системы SPP1000/XA и SPP1200/XA [43] являют собой пример ВС с массовым параллелизмом, память которых физически распределена между узлами, но логически она общая для всей вычислительной системы. Тем не менее большинство MPP-систем имеют как логически, так и физически распределенную память.

Благодаря свойству масштабируемости, MPP-системы являются сегодня лидерами по достигнутой производительности; наиболее яркий пример этому - Intel Paragon с 6768 процессорами. С другой стороны, распараллеливание в MPP-системах по сравнению с кластерами, содержащими немного процессоров, становится еще более трудной задачей. Следует помнить, что приращение производительности с ростом числа процессоров обычно вообще довольно быстро убывает (см. закон

Амдала). Кроме того, достаточно трудно найти задачи, которые сумели бы эффективно загрузить множество процессорных узлов. Сегодня не так уж много приложений могут эффективно выполняться на MPP-системе, Имеет место также проблема переносимости программ между системами с различной архитектурой. Эффективность распараллеливания во многих, случаях сильно зависит от деталей архитектуры MPP-системы, например топологии соединения процессорных узлов.

Самой эффективной была бы топология, в которой любой узел мог бы напрямую связаться с любым другим узлом, но в ВС на основе MPP это технически трудно реализуемо. Как правило, процессорные узлы в современных MPP-компьютерах образуют или двухмерную решетку (например, в SNI/Pyramid RM1000) или гиперкуб (как в суперкомпьютерах nCube [8]).

Поскольку для синхронизации параллельно выполняющихся процессов необходим обмен сообщениями, которые должны доходить из любого узла системы в любой другой узел, важной характеристикой является диаметр системы *D*. В случае двухмерной решетки *D* - sqrt(n), в случае гиперкуба *D* - 1n(n). Таким образом, при увеличении числа узлов более выгодна архитектура гиперкуба.

Время передачи информации от узла к узлу зависит от стартовой задержки и скорости передачи. В любом случае, за время передачи процессорные узлы успевают выполнить много команд, и это соотношение быстродействия процессорных узлов и передающей системы, вероятно, будет сохраняться — прогресс в производительности процессоров гораздо весомее, чем в пропускной способности каналов связи. Поэтому инфраструктура каналов связи в MPP-системах является объектом наиболее пристального внимания разработчиков.

Слабым местом MPP было и есть центральное управляющее устройство (ЦУУ) - при выходе его из строя вся система оказывается неработоспособной. Повышение надежности ЦУ У лежит на путях упрощения аппаратуры ЦУУ и/или ее дублирования.

Несмотря на все сложности, сфера применения ВС с массовым параллелизмом постоянно расширяется. Различные системы этого класса эксплуатируются во многих ведущих суперкомпьютерных центрах мира. Следует особенно отметить компьютеры Cray T3D и Cray T3E, которые иллюстрируют тот факт, что мировой лидер производства векторных суперЭВМ, компания Cray Research, уже не ориентируется исключительно на векторные системы. Наконец, нельзя не вспомнить, что суперкомпьютерный проект министерства энергетики США основан на MPP-системе на базе Pentium.

На рис. 14.14 показана структура MPP-системы RM1000, разработанной фирмой Pyramid.

В RM1000 используются микропроцессоры типа MIPS. Каждый узел содержит процессор R4400, сетевую карту Ethernet и два канала ввода/вывода типа SCSI. Реализованный вариант включает в себя 192 узла, но сеть соединений предусматривает масштабирование до 4096 узлов. Каждый узел имеет коммуникационный компонент для подключения к соединяющей сети, организованной по топологии двухмерной решетки. Связь с решеткой поддерживается схемами маршрутизации, с четырьмя двунаправленными линиями для связи с соседними узлами и одной линией для подключения к данному процессорному узлу. Скорость передачи информации в каждом направлении - 50 Мбит/с.

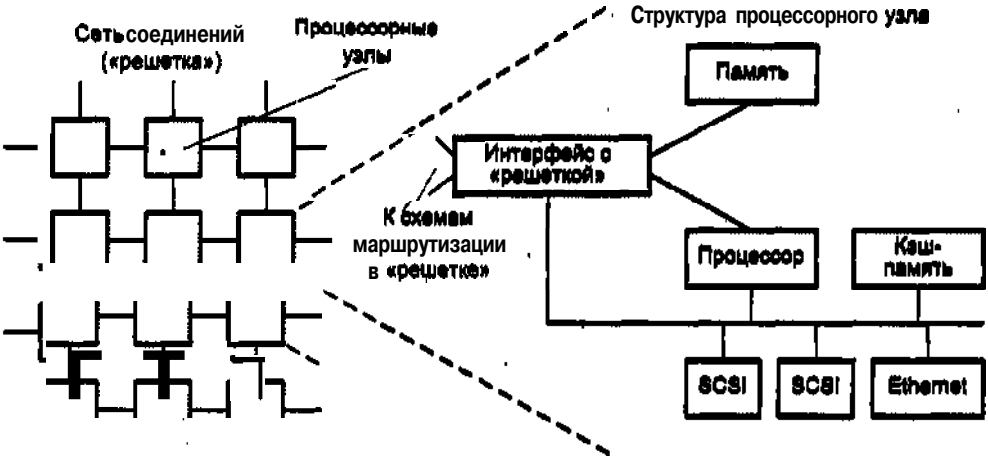


Рис. 14.14. Структура MPP-системы RM1000

Каждый узел работает под управлением своей копии операционной системы, управляет «своими»- периферийными устройствами и обменивается с другими узлами путем пересылки сообщений по сети соединений. Операционная система содержит средства для повышения надежности и коэффициента готрвности.

Отметим, что при создании MPP-систем разные фирмы отдают предпочтение различным микропроцессорам и топологиям сетей соединений (табл, 14.2).

Таблица 14.2. Основные характеристики некоторых MPP-систем

Фирма	Медаль	Тип микропроцессора	Организация соединений	Органиаация памяти	Наличие хост-компьютера
Intel	Paragon	I860	Двухмерная решетка	Распределенная	Нет
IBM	SP2	PowerPC 604e или Power2 SC	Коммутатор	Распределенная	Нет
Cray	Cray T3D	DEC Alpha	Трехмерный тор	Распределенная	Есть

Вычислительные системы с неоднородным доступом к памяти

Основные платформы, обычно применяемые при создании коммерческих мультипроцессорных систем, это SMP, MPP и кластеры. Наряду с ними в последнее время стали появляться решения, в которых акцентируется способ организации памяти. Речь идет о ВС, построенных в соответствии с технологией неоднородного доступа к памяти (NUMA, Non-Uniform Memory Access), точнее с кэш-когерентным доступом к неоднородной памяти (CC-NUMA).

В симметричных мультипроцессорных вычислительных системах (SMP) имеет место практический предел числа составляющих их процессоров. Эффективная схема с кэш-памятью уменьшает трафик шины между процессором и основной памятью, но по мере увеличения числа процессоров трафик шины также возрастает. Поскольку шина используется также для передачи сигналов, обеспечивающих когерентность, ситуация с трафиком еще более напрягается. С какого-то момента в плане производительности шина превращается в узкое место. Для систем типа SMP таким пределом становится число процессоров в пределах от 16 до 64. Например, объем SMP-системы Silicon Graphics Power Challenge ограничен 64 процессорами R10000, поскольку при дальнейшем увеличении числа процессоров производительность падает.

Ограничение на число процессоров в архитектуре SMP служит побудительным мотивом для развития кластерных систем, В последних же каждый узел имеет локальную основную память, то есть приложения «не видят» глобальной основной памяти. В сущности, когерентность поддерживается не столько аппаратурой, сколько программным обеспечением, что не лучшим образом сказывается на продуктивности. Одним из путей создания крупномасштабных вычислительных систем является технология CC-NUMA. Например, NUMA-система Silicon Graphics Origin поддерживает до 1024 процессоров R10000 [223], а Sequent NUMA-Q объединяет 252 процессора Pentium II [157].

На рис. 14.15 показана типичная организация систем типа CC-NUMA [36]. Имеется множество независимых узлов, каждый из которых может представлять собой, например, SMP-систему. Таким образом, узел содержит множество процессоров, у каждого из которых присутствуют локальные кэши первого (L1) и второго (L2) уровней. В узле есть и основная память, общая для всех процессоров этого узла, но рассматриваемая как часть глобальной основной памяти системы, В архитектуре CC-NUMA узел выступает основным строительным блоком. Например, каждый узел в системе Silicon Graphics Origin содержит два микропроцессора MIPS R10000, а каждый узел системы Sequent NUMA-Q включает в себя четыре процессора Pentium II. Узлы объединяются с помощью какой-либо сети соединений, которая представлена коммутируемой матрицей, кольцом или имеет иную топологию.

Согласно технологии CC-NUMA, каждый узел в системе владеет собственной основной памятью, но с точки зрения процессоров имеет место глобальная адресуемая память, где каждая ячейка любой локальной основной памяти имеет уникальный системный адрес. Когда процессор инициирует доступ к памяти и нужная ячейка отсутствует в его локальной кэш-памяти, кэш-память второго уровня (L2) процессора организует операцию выборки. Если нужная ячейка находится в локальной основной памяти, выборка производится с использованием локальной шины. Если же требуемая ячейка хранится в удаленной секции глобальной памяти, то автоматически формируется запрос, посылаемый по сети соединений на нужную локальную шину и уже по ней к подключенному к данной локальной шине кэшу. Все эти действия выполняются автоматически, прозрачны для процессора и его кэш-памяти,

В данной конфигурации главная забота - когерентность кэшей. Хотя отдельные реализации и. отличаются в деталях, общим является то, что каждый узел со-

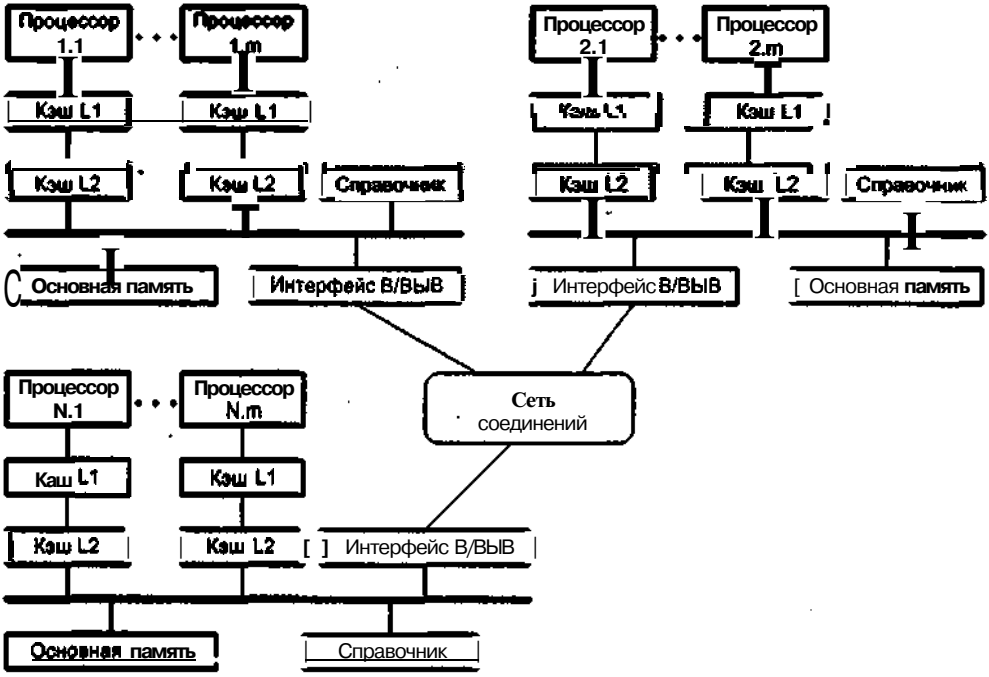


Рис. 14.15. Организация систем типа CC-NUMA

держит справочник, где хранится информация о местоположении в системе каждой составляющей глобальной памяти, а также о состоянии кэш-памяти.

Чтобы проанализировать, как работает такая схема, воспользуемся примером, приведенным в [182]. Пусть процессор 3 узла 2 (P2.3) запрашивает ячейку с адресом 798, расположенную в узле 1. Будет наблюдаться такая последовательность действий:

1. P2.3 выдает на шину наблюдения узла 2 запрос чтения ячейки 798.
2. Справочник узла 2 видит запрос и распознает, что нужная ячейка находится в узле 1.
3. Справочник узла 2 посылает запрос узлу 1, который принимается справочником узла 1.
4. Справочник основной памяти 1, действуя как заменитель процессора P2.3, запрашивает ячейку 798, так как будто он сам является процессором.
5. Основная память узла 1 реагирует тем, что помещает затребованные данные на локальную шину узла 1.
6. Справочник узла 1 перехватывает данные с шины.
7. Считанное значение через сеть соединений передается обратно в справочник узла 2.
8. Справочник узла 2 помещает полученные данные на локальную шину узла 2, действуя при этом как заместитель той части памяти, где эти данные фактически хранятся.

9. Данные перехватываются и передаются в кэш-память процессора P2.3 и уже оттуда попадают в процессор P2.3.

Из описания видно, как данные считываются из удаленной памяти с помощью аппаратных механизмов, делающих транзакции прозрачными для процессора. В основе этих механизмов лежит какая-либо форма протокола когерентности кэш-памяти. Большинство реализаций отличаются именно тем, какой именно протокол когерентности используется.

Вычислительные системы на базе транспьютеров

Появление транспьютеров связано с идеей создания различных по производительности ВС (от небольших до мощных массивно-параллельных) посредством прямого соединения однотипных процессорных чипов. Сам термин объединяет два понятия — «транзистор» и «компьютер».

Транспьютер - это сверхбольшая интегральная микросхема (СБИС), заключающая в себе центральный процессор, блок операций с плавающей запятой (за исключением транспьютеров первого поколения T212 и T414), статическое оперативное запоминающее устройство, интерфейс с внешней памятью и несколько каналов связи. Первый транспьютер был разработан в 1986 году фирмой Inmos.

Канал связи состоит из двух последовательных линий для двухстороннего обмена Он позволяет объединить транспьютеры между собой и обеспечить взаимные коммуникации. Данные могут пересылаться поэлементно или как вектор. Одна из последовательных линий используется для пересылки пакета данных, а вторая — для возврата пакета подтверждения, который формируется как только пакет данных достигнет пункта назначения.

На базе транспьютеров легко могут быть построены различные виды ВС. Так, четыре канала связи обеспечивают построение двухмерного массива, где каждый транспьютер связан с четырьмя ближайшими соседями. Возможны и другие конфигурации, например объединение транспьютеров в группы с последующим соединением групп между собой. Если группа состоит из двух транспьютеров, для подключения ее к другим группам свободными остаются шесть каналов связи (рис. 14.16, *а)*. Комплекс из трех транспьютеров также оставляет свободными шесть каналов (рис. 14.16,*б*), а для связи с «квартетом» транспьютеров остаются еще четыре канала связи (рис. 14.16, *в)*. Группа из пяти транспьютеров может иметь полный набор взаимосвязей, но за счет потери возможности подключения к другим группам.

Особенности транспьютеров потребовали разработки для них специального языка программирования *Occam*. Название языка связано с именем философа-схоласта четырнадцатого века Оккама - автора концепции «бритвы Оккамак «entia praeter necessitatem non sunt multiplicanda» — «понятия не должны умножаться без необходимости». Язык обеспечивает описание простых операций пересылки данных между двумя точками, а также позволяет явно указать на параллелизм при выполнении программы несколькими транспьютерами. Основным понятием программы на языке Occam является *процесс,* состоящий из одного или более опера-

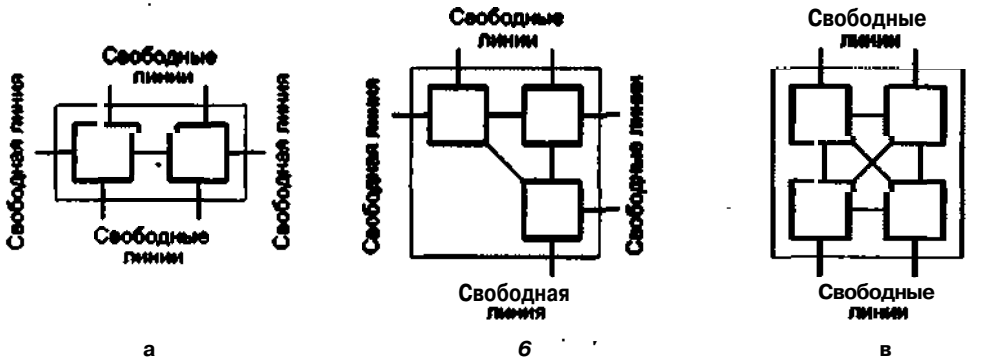


Рис. 14.16. Группы из полностью взаимосвязанных транспьютеров: а — два транспьютера; *б*- три транспьютера; в - четыре транспьютера

торов программы, которые могут быть выполнены последовательно или параллельно. Процессы могут быть распределены по транспьютерам вычислительной системы, при этом оборудование транспьютера поддерживает совместное использование транспьютера одним или несколькими процессами.

Принято говорить о двух поколениях транспьютеров и языка Occam. Первое поколение отражает требования тех приложений, для которых транспьютеры и разрабатывались: цифровой обработки сигналов и систем реального времени. Для подобных задач нужны сравнительно небольшие ВС со скоростными каналами связи (главным образом, между соседними процессорами) и быстрым переключением контекста. Под *контекстом* понимается содержимое регистров, которое при переходе к новой задаче в ходе многозадачной обработки может быть изменено и поэтому должно быть сохранено, а при возврате к старой задаче — восстановлено. Многомашинные ВС, построенные на транспьютерах первого поколения (T212, T414 и T805), по своей производительности были сравнимы с другими типами ВС того времени.

С появлением вычислительных систем второго и третьего поколений стало ясно, что ВС на транспьютерах ранней организации уже стали неконкурентоспособными, что и побудило к созданию их второго поколения (T9000). В последних существенно повышена производительность и улучшены каналы связи. Главная особенность транспьютеров второго поколения — развитые коммуникационные возможности, хотя в вычислительном плане, даже несмотря на наличие в них блоков для операций с плавающей запятой, они сильно уступают универсальным микропроцессорам, таким как PowerPC и Pentium.

Архитектура транспьютера

Обобщенная структура транспьютера, показанная на рис. 14.17, включает в себя:

- центральный процессор;
- АЛУ для операций с плавающей запятой;
- каналы связи;
- внутреннюю память (ОЗУ);

- интерфейс для подключения внешней памяти;
- интерфейс событий (систему прерываний);
- логику системного сервиса (систему обслуживания);
- таймеры.

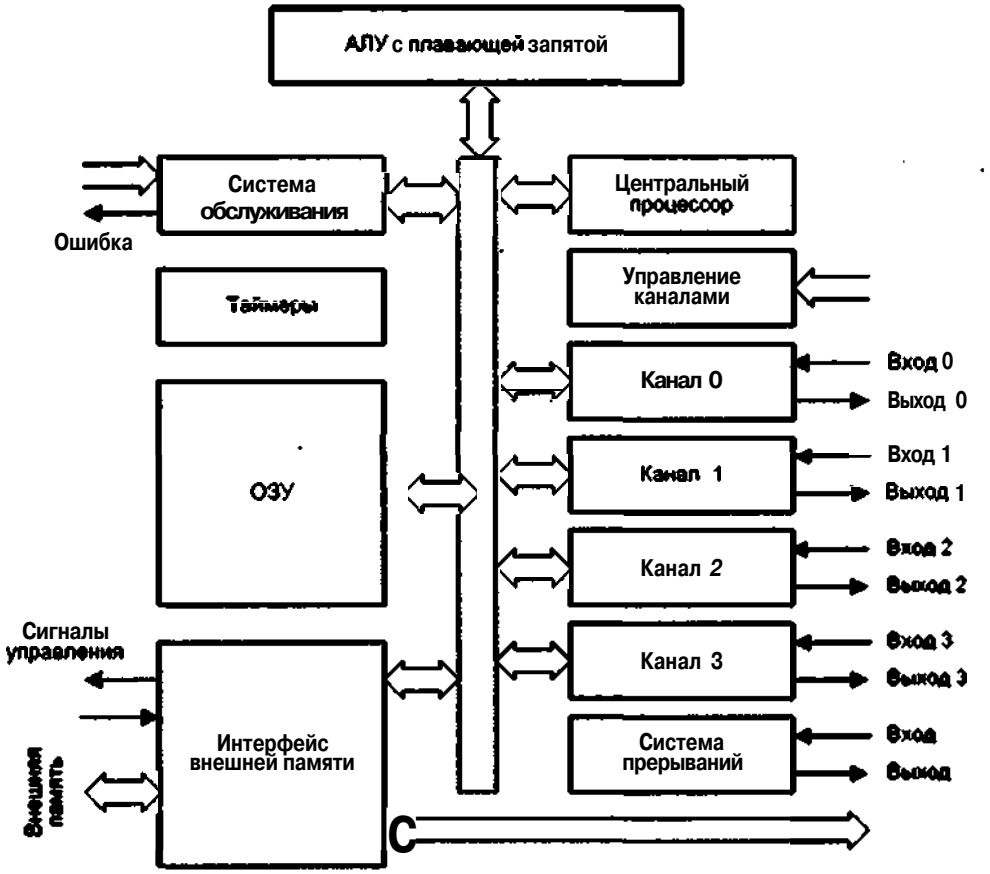


Рис. 14.17. Базовая внутренняя архитектура транспьютера

Первый транспьютер Т212 содержал 16-разрядный арифметический процессор. Последующие транспьютеры были оснащены 32-разрядным целочисленным процессором (Т414,1985) и процессором с плавающей запятой (Т800, Т9000), дающим существенное повышение скорости вычислений (до 100 MIPS). Версии, поддерживающие процессор с ПЗ, организованы так, что этот процессор и целочисленный процессор могут работать одновременно. В дополнение, в Т9000 добавлена внутренняя кэш-память и процессор виртуального канала. Сам по себе процессор транспьютера построен по архитектуре RISC, имеет микропрограммное УУ, а команды в нем выполняются за минимальное число циклов процессора. Простые операции, такие как сложение или вычитание, занимают один цикл, в то время как более сложные операции требуют нескольких циклов. Команды состоят

из одного или нескольких байтов. Большинство версий транспьютеров имеют по четыре последовательных какала связи со скоростью передачи по каналу порядка 10 Мбит/с. По мере развития транспьютеров повысилась скорость передачи по каналам связи. Емкость внутренней памяти (вначале 2 Кбайт) также возросла. Появилась возможность подключения внешней памяти через интерфейс памяти. Схема этого интерфейса программируется и способна формировать различные сигналы для удовлетворения различных требований самых разнообразных микросхем внешней памяти.

Передача информации производится синхронно под воздействием либо общего генератора тактовых импульсов (ГТИ), либо локальных ГТИ с одинаковой частотой следования импульсов. Информация передается в виде пакетов. Каждый раз, когда пересылается *пакет данных,* приемник отвечает пакетом *подтверждения* (рис. 14.18).

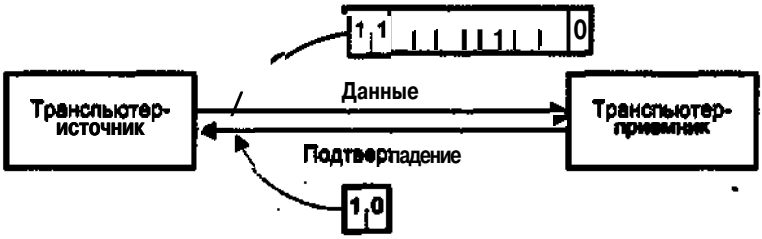


Рис. 14.18. Организация ввода/вывода в транспьютерной системе

Пакет данных состоит из двух битов-единиц, за которыми следуют 8-битовые данные и ноль (всего 11 бит). Пакет подтверждения - это простая комбинация 10 (всего два бита), она может быть передана, как только пакет данных будет идентифицирован интерфейсом входного канала. Каналы обеспечивают аппаратную поддержку операторов ввода и вывода языка Occam и функционируют словно каналы ПДП, то есть пакеты могут пересылаться один за другим как векторы. Для коммуникаций между процессами внутри транспьютера вместо внешних каналов операторы ввода/вывода используют внутренние каналы транспьютера.

Интерфейс событий дает возможность внешнему устройству привлечь внимание и получить подтверждение. Этот интерфейс функционирует как входной канал и аналогично программируется.

Вычислительные системы с обработкой по принципу волнового фронта

Интересной разновидностью систолических структур являются *матричные процессоры волнового фронта* (wavefront array processor), иногда называемые также *волновыми* или *фронтальными*.

Как уже отмечалось, в основе построения систолических ВС лежит глобальная синхронизация массива процессоров, предусматривающая наличие сети распределения синхронизирующих сигналов по всей структуре. Б системах с очень большим числом ПЭ начинает сказываться запаздывание тактовых сигналов. Послед-

нее обстоятельство особенно ощутимо при исполнении массива на базе СБИС, где связи между ПЭ очень тонкие физически, вследствие чего обладают повышенной емкостью. В итоге возникают серьезные проблемы с синхронизацией, для устранения которых предпочтительным представляется использование самосинхронизирующихся схем управления процессорными элементами. *Самосинхронизация* заключается в том, что моменты начала очередной операции каждый ПЭ определяет автоматически, по мере готовности соответствующих операндов. В итоге отпадает необходимость глобальной синхронизации, исчезают непроизводительные временные издержки и повышается общая производительность всей структуры, хотя и усложняется аппаратная реализация каждого ПЭ [135,147].

Волновые процессорные массивы сочетают систолическую конвейерную обработку данных с асинхронным характером потока данных. В качестве механизма координации межпроцессорного обмена в волновых системах принята асинхронная *процедура связи с подтверждением* (handshake). Когда какой-либо процессор

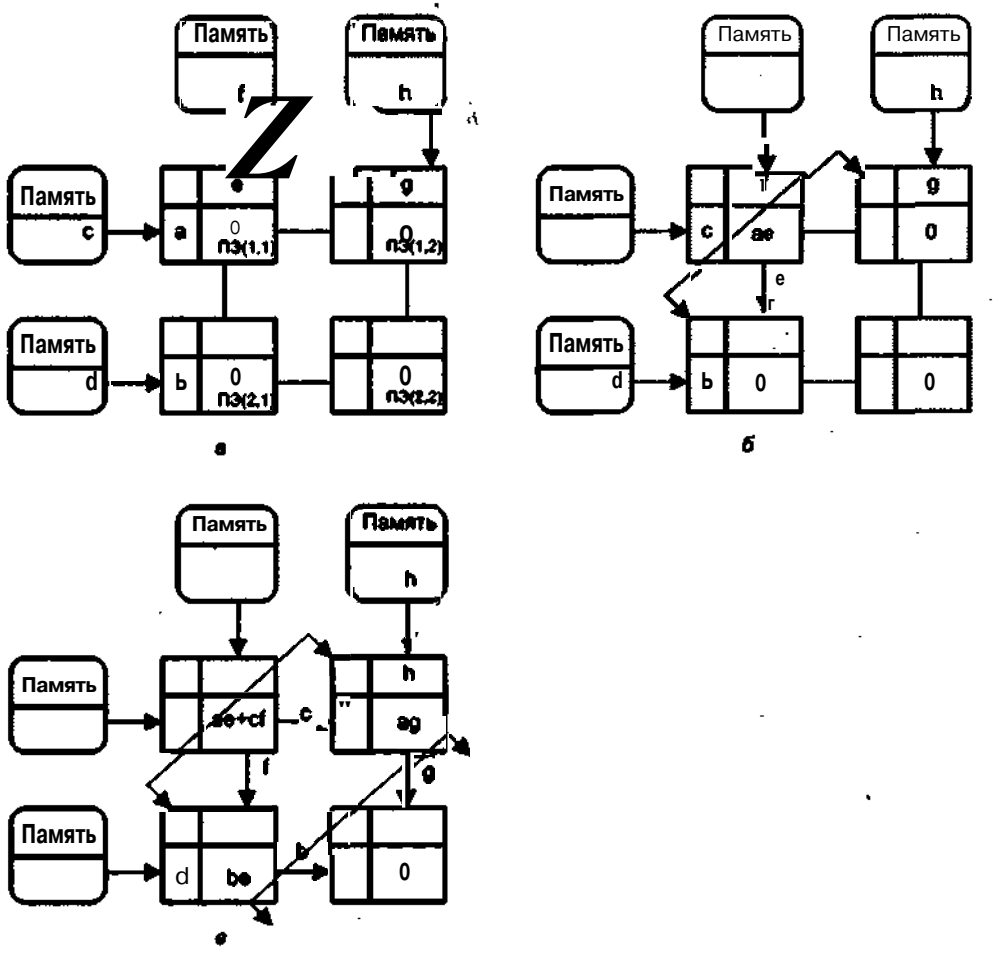


Рис. 14.19. Выполнение матричного умножения на волновой вычислительной системе

массива завершает свои вычисления и готов передать данные соседу, он может это сделать, лишь когда последний будет готов к их приему. Для проверки готовности соседа передающий процессор сначала направляет ему запрос, а данные посылает только после получения подтверждения о готовности их принять. Такой механизм обеспечивает соблюдение заданной последовательности вычислений и делает прохождение фронта вычислений через массив плавным, причем задача соблюдения последовательности вычислений решается непосредственно, в то время как в систолических ВС для этого требуется строгая синхронизация.

Концепцию массива процессоров волнового фронта проиллюстрируем на примере матричного умножения (рис. 14.19).

Вычислительная система в примере состоит из процессорных элементов, имеющих на каждом входе данных буфер на один операнд. Всякий раз, когда буфер пуст, а в памяти, являющейся источником данных, содержится очередной операнд, производится немедленное его считывание в буфер соответствующего процессора. Операнды из других ПЭ принимаются на основе протокола связи с подтверждением.

Рисунок 14.19, *а* фиксирует ситуацию после первоначального заполнения входных буферов. Здесь ПЭ(1,1) суммирует произведение *а* х *е* с содержимым своего аккумулятора и транслирует операнды *а и е* своим соседям. Таким образом, первый волновой фронт вычислений (см. рис. 14.19,*б)* перемещается в направлении от ПЭ(1,1) к П Э(1,2) и ПЭ(2,1). Рисунок 14.19, в иллюстрирует продолжение распространения первого фронта и исход от ПЭ(1,1) второго фронта вычислений.

По сравнению с систолическими ВС массивы волнового фронта обладают лучшей масштабируемостью, проще в программировании и характеризуются более высокой отказоустойчивостью.

Контрольные вопросы

1. По какому признаку вычислительную систему можно отнести к сильно связанным или слабо связанным ВС?
2. Какие уровни параллелизма реализуют симметричные мультипроцессорные системы?
3. Какими средствами поддерживается когерентность кэш-памяти в SMP-системах?
4. Оцените достоинства и недостатки различных SMP-архитектур.
5. В чем состоит принципиальное различие между матричными и симметричными мультипроцессорными вычислительными системами?
6. Какие две проблемы призвана решить кластерная организация вычислительной системы?
7. Существуют ли ограничения на число узлов в кластерной ВС? И если существуют, то чем они обусловлены?
8. Какие задачи в кластерной вычислительной системе возлагаются на специализированное (кластерное) программное обеспечение?

9. Каким образом может быть организовано взаимодействие между узлами кластерной ВС?
10. При каком количестве процессоров ВС можно отнести к системам с массовой параллельной обработкой?
11. Как организуется координация процессоров и распределение между ними заданий в MPP-системах?
12. Какие топологии можно считать наиболее подходящими для MPP-систем и почему?
13. Поясните назначение справочника в вычислительных системах типа CC-NUMA.
14. Какие протоколы когерентности, на ваш взгляд, наиболее подходят для ВС, построенных на технологии CC-NUMA?
15. Какие черты транспьютера отличают его от стандартной однокристальной ВМ?
16. Какими аппаратными и программными средствами поддерживается взаимодействие соседних транспьютеров в вычислительной системе?
17. Сколько линий поддерживает канал связи транспьютера, как они используются и в каком режиме осуществляется ввод/вывод?
18. Какие особенности транспьютеров облегчает реализовать язык Occam?
19. Опишите структуру пакета данных и пакета подтверждения, передаваемых в транспьютерных ВС.
20. Какие из рассмотренных типов вычислительных систем могут быть построены на базе транспьютеров и в каких случаях это наиболее целесообразно?
21. В чем состоят сходство и различие между систолическими ВС и вычислительными системами с обработкой по принципу волнового фронта?
22. Как организуется межпроцессорный обмен в массивах волнового фронта?

Глава 15

Потоковые и редукционные вычислительные системы

В традиционных ВМ команды в основном выполняются в естественной последовательности, то есть в том порядке, в котором они хранятся в памяти. То же самое можно сказать и о традиционных многопроцессорных системах, где одновременно могут выполняться несколько командных последовательностей, но также в порядке размещения каждой из них в памяти. Это обеспечивается наличием в каждом процессоре счетчика команд. Выполнение команд в каждом процессоре — поочередное и потому достаточно медленное. Для получения выигрыша программист или компилятор должны определить независимые команды, которые могут быть поданы на отдельные процессоры, причем так, чтобы коммуникационные издержки были не слишком велики.

Традиционные (фон-неймановские) вычислительные системы, управляемые с помощью программного счетчика, иногда называют *вычислительными системами, управляемыми последовательностью команд* (control flow computer). Данный термин особенно часто применяется, когда нужно выделить этот тип ВС из альтернативных типов, где последовательность выполнения команд определяется не центральным устройством управления со счетчиком команд, а каким-либо иным способом. Если программа, состоящая из команд, хранится в памяти, возможны следующие альтернативные механизмы ее исполнения:

- команда выполняется, после того как выполнена предшествующая ей команда последовательности;
- команда выполняется, когда становятся доступными ее операнды;
- команда выполняется, когда другим командам требуется результат ее выполнения.

Первый метод соответствует традиционному механизму с управлением последовательностью команд; второй механизм известен *как управляемый данными* (data driven) или *потоковый* (dataflow); третий метод называют механизмом *управления по запросу* (demand driven).

Общие идеи нетрадиционных подходов к организации вычислительного процесса показаны на рис. 15.1, а их более детальному изложению посвящен текущий раздел.

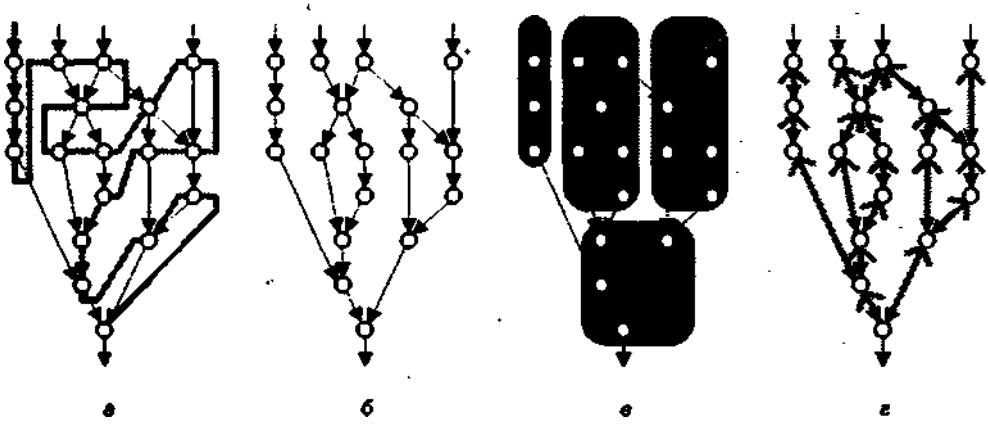


Рис. 15.1. Возможные вычислительные модели: *а* - фон-неймановская; *б*- потоковая; *в* — макропотоковая; г—редукционная

Вычислительные системы с управлением вычислениями от потока данных

Идеология вычислений, управляемых потоком данных (потоковой обработки), была разработана в 60-х гадах Карпом и Миллером. В начале 70-х годов Деннис, а позже и другие начали разрабатывать компьютерные архитектуры, основанные на вычислительной модели с потоком данных.

Вычислительная модель потоковой обработки

В потоковой вычислительной модели для описания вычислений используется ориентированный граф, иногда называемый *графом потоков данных* (dataflow graph). Этот граф состоит из *узлов* или *вершин,* отображающих операции, *и ребер* или *дуг,* показывающих потоки данных между теми вершинами графа, которые они соединяют.

Узловые операции выполняются, когда по дугам в узел поступила вся необходимая информация. Обычно узловая операция требует одного или двух операндов, а для условных операции необходимо наличие входного логического значения. По выполнении операции формируются один или два результата. Таким образом, у каждой вершины может быть от одной до трех входящих дуг и одна или две выходящих. После активации вершины и выполнения узловой операции (это иногда называют *инициированием вершины)* результаты передаются по ребрам к ожидающим вершинам. Процесс повторяется, пока не будут инициированы все вершины и получен окончательный результат. Одновременно может быть инициировано несколько узлов, при этом параллелизм в вычислительной модели выявляется автоматически.

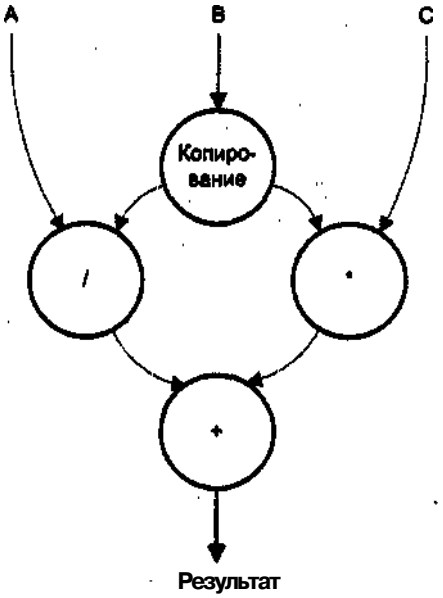


Рис. 15.2. Граф потоков данных для выражения A/B + B x C

На рис. 15.2 показан простой потоковый граф для вычисления выражения $f=A/B+B\times C$. Входами служат переменные A, B и C, изображенные вверху графа. Дуги между вершинами показывают тракты узловых операций. Направление вычислений — сверху вниз. Используются три вычислительные операции: сложение, умножение и деление. Заметим, что B требуется в двух узлах. Вершина «Копирование»- предназначена для формирования дополнительной копии переменной B.

Данные (операнды/результаты), перемещаемые вдоль дуг, содержатся в опознавательных информационных кадрах, маркерах специального формата — *«токенах»* (иначе «фишках»- или маркерах доступа). Рисунок 15.3 иллюстрирует движение токенов между узлами. После поступления на граф входной информации маркер, содержащий значение A, направляется в вершину деления; токен с переменной B — в вершину копирования; токен с переменной C — в вершину умножения. Активирована может быть только вершина «Копирование», поскольку у нее лишь один вход и на нем уже присутствует токен. Когда токены из вершины «Копирование*' будут готовы, узлы умножения и деления также получат все необходимые маркеры доступа и могут быть инициированы. Последняя вершина ждет завершения операций умножения и деления, то есть когда на ее входе появятся все необходимые токены.

Практические вычисления требуют некоторых дополнительных возможностей, например при выполнении команд условного перехода. По этой причине в потоковых графах предусмотрены вершины-примитивы нескольких типов (рис. 15.4):

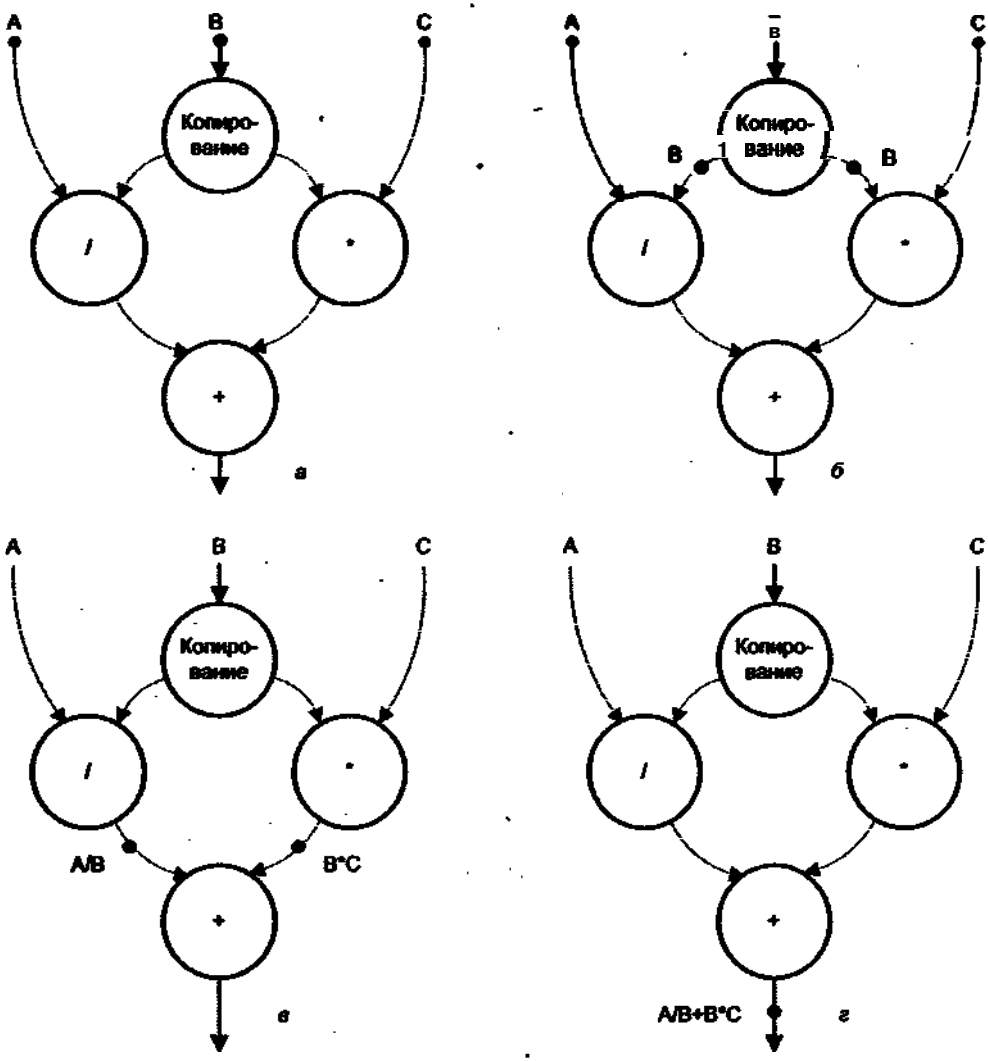


Рис. 15.3. Движение маркеров при вычислении A/B + B x C: а - после подачи входных данных; б - после копирования; *в* - после умножения и деления; г - после суммирования

-*двухвходовая операционная вершина* — узел с двумя входами и одним выходом. Операции производятся над данными, поступающими с левой и правой входных дуг, а результат выводится через выходную дугу;

- *одновходовая операционная вершина* — узел с одним входом и одним выходом. Операции выполняются над входными данными, результат выводится через выходное ребро;

- *вершина ветвления* — узел с одним входом и двумя выходами. Осуществляет копирование входных данных и их вывод через две выходные дуги. Путем комбинации таких узлов можно строить вершины ветвления на *m* выходов;

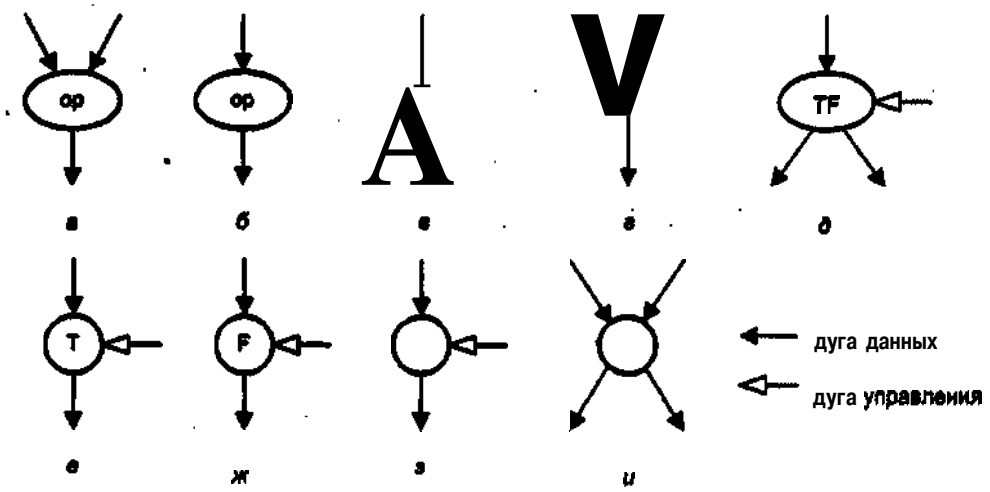


Рис. 15.4. Примитивы узлов: *а* - двухвходовая операционная вершина; *б* - одновходовая операционная вершина; *в* — вершина ветвления; г— вершина слияния; д — TF-коммутатор; е — Т-коммутатор; ж — F-коммутатор; *з* - вентиль; *и* — арбитр

- *вершина слияния* — узел с двумя входами и одним выходом. Данные поступают только с какого-нибудь одного из двух входов. Входные данные без изменения подаются на выход. Комбинируя такие узлы, можно строить вершины слияния с *m* входами;
- *вершина управления* — существует в перечисленных ниже трех вариантах:
  - *TF-коммутатор* — узел с двумя входами и одним выходом. Верхний вход — это дуга данных, а правый - дуга управления (логические данные). Если значение правого входа истинно (T — True), то входные данные выводятся через левый выход, а при ложном значении на правом входе (F — False) данные следуют через правый выход;
  - D *вентиль* — узел с двумя входами и одним выходом. Верхний вход — дуга данных, а правый — дуга управления. При истинном значении на входе управления данные выводятся через выходную дугу;
  - *арбитр* — узел с двумя входами и двумя выходами. Все дуги являются дугами данных. Первые поступившие от двух входов данные следуют через левую дугу, а прибывшие впоследствии — через правую выходную дугу. Активация вершины происходит в момент прихода данных с какого-либо одного входа.

Процесс обработки может выполняться аналогично конвейерному режиму: после обработки первого набора входных сигналов на вход графа может быть подан второй и т. д. Отличие состоит в том, что промежуточные результаты (токены) первого вычисления не обрабатываются совместно с промежуточными результатами второго и последующих вычислений. Результаты обычно требуются в последовательности использования входов.

Существует множество случаев, когда определенные вычисления должны повторяться *с* различными данными, особенно в программных циклах. Программ-

мые циклические процессы в типовых языках программирования могут быть воспроизведены путем подачи результатов обратно на входные узлы. При формировании итерационного кода часто применяются переменные цикла, увеличиваемые после каждого прохода тела цикла. Последний завершается, когда переменная цикла достигает определенного значения. Метод применим и при потоковой обработке, как это показано на рис. 15.5.

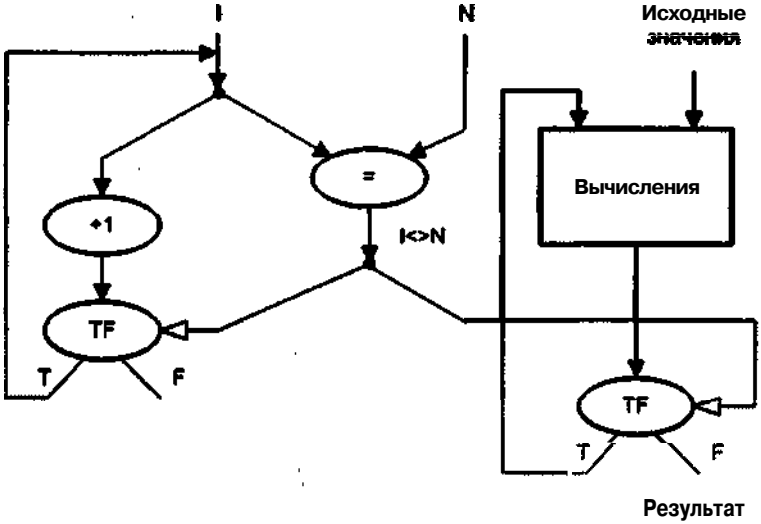


Рис. 15.5. Циклы при потоковой обработке

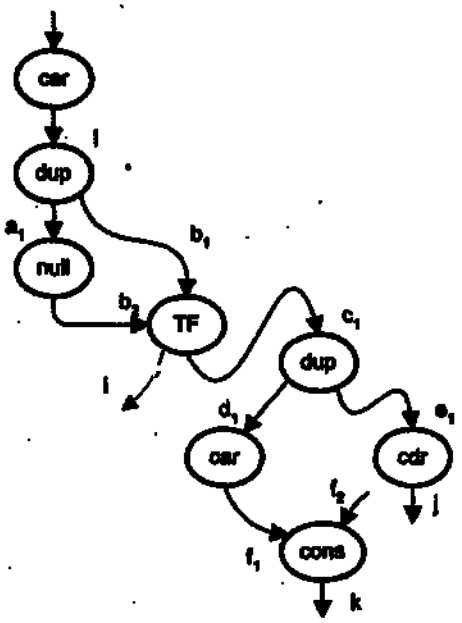
Все известные потоковые вычислительные системы могут быть отнесены к двум основным типам: *статическим* и *динамическим.* В свою очередь, динамические потоковые ВС обычно реализуются по одной из двух схем: *архитектуре с помеченными токенами и архитектуре с явно адресуемыми токенами.*

Архитектура потоковых вычислительных систем

В потоковых ВС программа вычислений соответствует потоковому графу, который хранится в памяти системы в виде таблицы. На рис. 15.6 показаны пример графа потоковой программы и содержание адекватной ему таблицы [2]. •

Принципиальная схема потоковой вычислительной системы (рис. 15.7) включает в себя блок управления (CS), где хранится потоковый граф, который используется для выборки обрабатываемых команд, а также функциональный блок (FS), выполняющий команду, переданную из CS, и возвращающий результат ее выполнения в CS.

Блоки CS и FS работают асинхронно и параллельно, обмениваясь многочисленными *пакетами команд* и результатами их выполнения. В *пакете результата,* поступающем из блока FS, содержится значение результата (val) и адрес команды, для которой пакет предназначен (des). На основании этого адреса блок CS проверяет возможность обработки команды. Команда может быть одноперандной или двухоперандной. В последнем случае необходимо подтверждение наличия обоих



opc - код операции:
opr1, opr2 - поля операндов;
des - адрес назначения:
tag - тег поступления данных (2 бита)

	opc	opr1	opr2	des	tag
	car			1	
\	dup			a_1, b_1	0.1
a	null			b_2	0.1
b	TF			i, c_1	0.1
c	dup			d_1, e_1	0.1
d	car			f_1	0.1
e	cdr			i	0, 1
f	cons			k	0.1

а б

Рис. 1 5 . 6 . Пример формы хранения потоковой программы: а - потоковый граф; *б* - память функционального блока

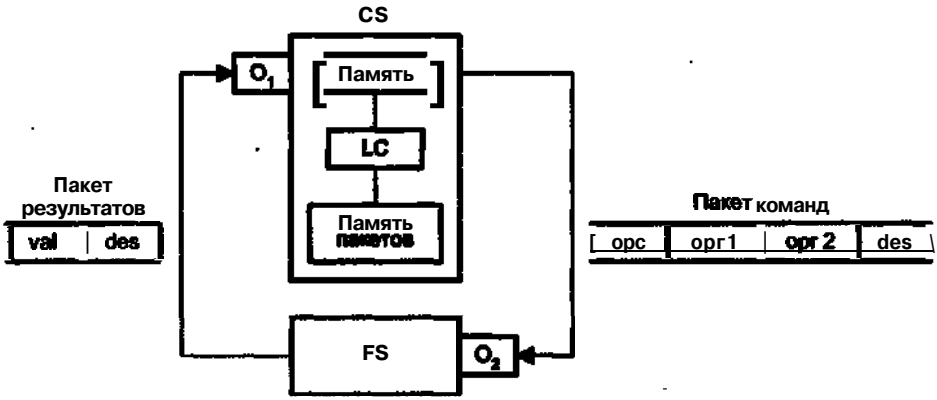


Рис. 1 5 . 7 . Структура потоковой вычислительной системы

операндов (орг 1 и орг 2), и для этого устанавливается специальный признак. Блок управления загрузкой (LC) каждый раз при активировании определенной функции загружает из памяти программ код этой функции.

Для повышения степени параллелизма блоки CS и F5 строятся по модульному принципу, а графы потоковой программы распределяются между модулями с помощью мультиплексирования [89].

Статические потоковые вычислительные системы

*Статическая потоковая архитектура, известная т*а к *же под назв*а н и*ем «еди* нстве ный-токен-на-дугу» (single-token-per-arc dataflow), была предложена Деннисом в 1975 году [89]. В ней допускается присутствие на ребре графа не более чем одного токена. Это выражается в правиле активации узла [90]: вершина активируется, когда на всех ее входных дугах присутствует по токену и ни на одном из ее выходов токенов нет. Для указания вершине о том, что ее выходной токен уже востребован последующим узлом (узлами) графа, в ВС обычно прибегают к механизму подтверждения с квитированием связи, как это показано на рис. 15.8. Здесь процессорами в ответ на инициирование узлов графа посылаются специальные контрольные токены.

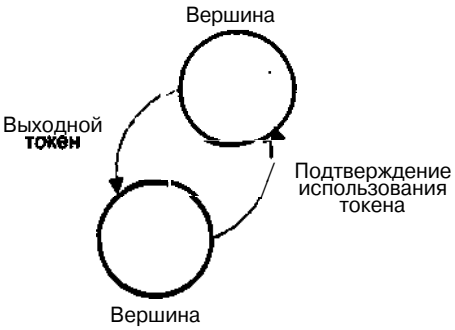


Рис. 15.8. Механизм подтверждения с квитированием

Типовую статическую потоковую архитектуру иллюстрирует рис 15.9.

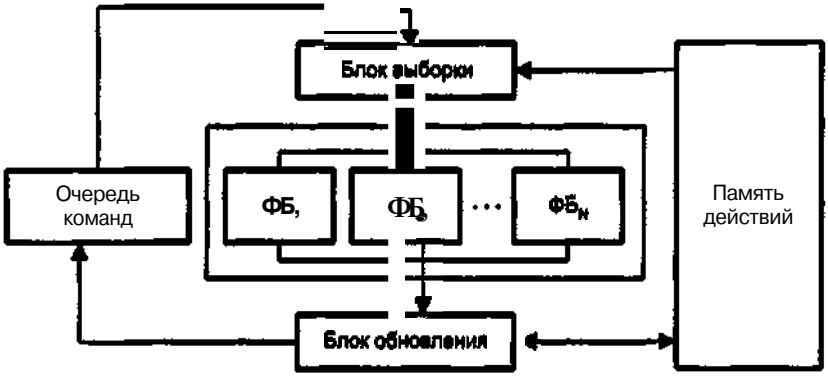


Рис. 1 5 . 9 , Структура процессорного элементе типовой статической потоковой системы

Память действий (узел потоковой машины, хранящий потоковый граф и циркулирующий на нем поток данных) состоит их двух блоков памяти: команд/данных и управляющей памяти. Вершина графа представлена в памяти команд/данных кадром, содержащим следующие поля:

- код операции;
- операнд 1;
- ...

- операнд*N;*
- вершина/дуга 1;
- ...
- вершина/дуга K.

Каждому кадру в памяти команд/данных соответствует кадр в управляющей памяти, содержащий биты наличия для всех операндов и всех полей «вершина/дуга». Бит наличия операнда устанавливается в единицу, если этот операнд доступен, то есть если токен, содержащий данный операнд, уже поступил по входной дуге графа. Для поля «вершина/дуга» установленный бит наличия означает, что выходная дуга, ассоциированная с данным полем, не содержит токена. Вершина графа, описанная в памяти команд/данных, может быть активирована (операция может быть выполнена), если все биты наличия в соответствующем кадре памяти управления установлены в единицу. Когда данная ситуация распознается блоком обновления, он помещает пакет команды в очередь команд. Опираясь на очередь команд и содержимое памяти действий, блок выборки составляет пакет операции и направляет его в один из функциональных блоков. После выполнения требуемой операции функциональный блок создает пакет результата и передает его в блок обновления, который в соответствии с полученным результатом изменяет содержимое памяти действий.

Основное преимущество рассматриваемой модели потоковых вычислений заключается в упрощенном механизме обнаружения активированных узлов. К сожалению, статическая модель обладает множеством серьезных недостатков [55]. Во-первых, данный механизм не допускает параллельного выполнения независимых итераций цикла. Другой нежелательный эффект - колебания трафика токенов. Наконец, в современных языках программирования отсутствует поддержка описанного режима обработки данных. Несмотря на это, несколько образцов статических ВС все-таки были созданы или, по крайней мере, спроектированы.

Архитектуру первой системы, строго соответствующей модели потоковых вычислений типа «единственный-токен-на-дугу», предложил Деннис (рис. 15.10). По изначальной идее, система представляла собой кольцо из процессорных элементов и элементов памяти, в котором информация передается в форме пакетов.

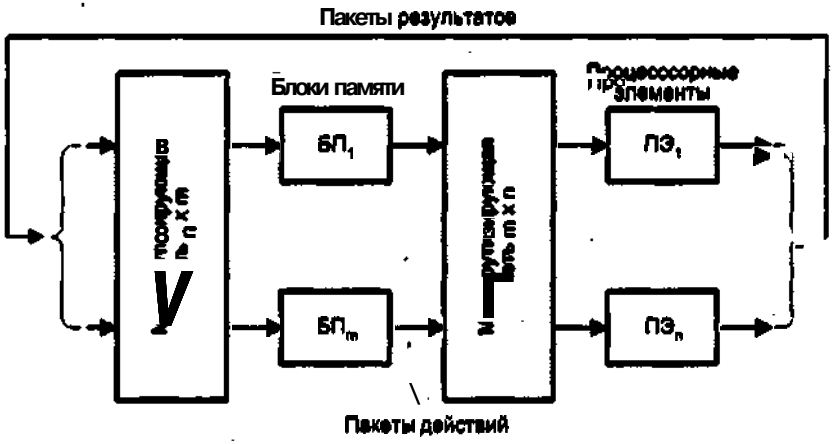


Рис. 15.10. Статическая потоковая архитектура по Деннису

Процессорные элементы должны были получать так называемые *пакеты действий* (activity templates) в виде: Код операции • Операнды • Адресат. Здесь точка обозначает операцию составления целого слова из его частей, «Код операции» определяет подлежащую выполнению операцию, «Операнды» — используемые в операции числа, а «Адресат» — указывает место, куда должен быть направлен результат операции. Отметим, что в полях операндов задаются не адреса ячеек памяти, а непосредственно числа, которые должны участвовать в операции. Это имеет свои преимущества и недостатки. Первое состоит в том, что в каждый момент времени операнды могут быть использованы только одной выбранной вершиной. Изъян идеи — невозможность включения в команду в качестве операндов сложных структур данных, даже простых векторов и массивов.

Пакет результата имеет вид: Значение • Адресат, где в поле «Значение» содержится значение результата, полученное после выполнения операции. Эти пакеты передаются по маршрутизирующей сети в так называемые *ячейки команды* блока памяти, а именно в указанные в их поле «Адресат». Когда получены все входные пакеты (токены), ячейка команды порождает пакет операции. Обычно для генерации пакета операции ячейке команды нужны два входных пакета с операндами. Затем пакет операции маршрутизируется к одному из процессорных элементов (ПЭ). Если все процессорные элементы идентичны (гомогенная система), может быть выбран любой свободный ПЭ. В негомогенных системах со специализированными ПЭ, способными выполнять только определенные функции, выбор нужного ПЭ производится по коду операции, заключенном в пакете операции.

В несколько усовершенствованном варианте рассмотренная ВС была спроектирована под руководством того же Денниса в Массачусетсском технологическом институте (MIT). Система состояла из пяти основных подсистем: секции памяти, секции процессоров, сети арбитража, сети управления и сети распределения. Все коммуникации между подсистемами осуществлялись путем асинхронной передачи пакетов по каналам. Прототип ВС содержал восемь процессорных элементов на базе микропроцессоров и сеть маршрутизации пакетов, построенную на маршрутизирующих элементах 2x2.

В качестве других примеров статических потоковых ВС можно упомянуть: LAU System [80,81], TI's Distributed Data Processor [82], DDMI Utah Data Driven Machine [85], NEC Image Pipelined Processor [73], Hughes Dataflow Multiprocessor [218].

Динамические потоковые вычислительные системы

Производительность потоковых систем существенно возрастает, если они в состоянии поддерживать дополнительный уровень параллелизма, соответствующий одновременному выполнению отдельных итераций цикла или параллельной обработке пар элементов в векторных операциях. Кроме того, в современных языках программирования активно используются так называемые *реентерабельные процедуры,* когда в памяти хранится только одна копия кода процедуры, но эта копия является повторно входимой (реентерабельной). Это означает, что к процедуре можно еще раз обратиться, не дожидаясь завершения действий в соответствии с предыдущим входом в данную процедуру. Отсюда желательно, чтобы все обращения к реентерабельной процедуре также обрабатывались параллельно. Задача обеспечения дополнительного уровня параллелизма решается в динамических по-

токовых ВС и реализуется двумя вариантами архитектуры потоковой ВС: *архитектуры с помеченными токенами и архитектуры с явно адресуемыми токенами*.

Архитектура потоковых систем с помеченными токенами

В архитектуре с помеченными токенами (tagged-token architecture) память хранит один экземпляр потокового графа. Каждый токен содержит *тег* (фишку), состоящий из адреса команды, для которой предназначено заключенное в токене значение, и другой информации, определяющей вычислительный контекст, в котором данное значение используется, например номера итерации цикла. Этот контекст называют «цветом значения», а токен соответственно называют «окрашенным»-, в силу чего метод имеет еще одно название — *метод окрашенных токенов*. Каждая дуга потокового графа может рассматриваться как вместилище, способное содержать произвольное число токенов с различными тегами. Основное правило активирования вершины в динамических потоковых ВС имеет вид: *вершина активируется, когда на всех ее входных дугах присутствуют токены с идентичными тегами*.

Типовая архитектура потоковой системы с помеченными токенами показана на рис. 15.11. Для обнаружения одинаково окрашенных токенов (токенов с одинаковыми тегами) в конвейер процессорного элемента введен согласующий блок. Этот блок получает очередной токен из очереди токенов и проверяет, нет ли в памяти согласования его партнера (токена с идентичным тегом). Если такой партнер не обнаружен, принятый токен заносится в память согласования. Если же токен-партнер уже хранится в памяти, то блок согласования извлекает его оттуда и направляет оба токена с совпавшими тегами в блок выборки. На основе общего тега блок выборки находит в памяти команд/данных соответствующую команду и формирует пакет операции, который затем направляет в функциональный блок. Функциональный блок выполняет операцию, создает токены результата и помещает их в очередь токенов.

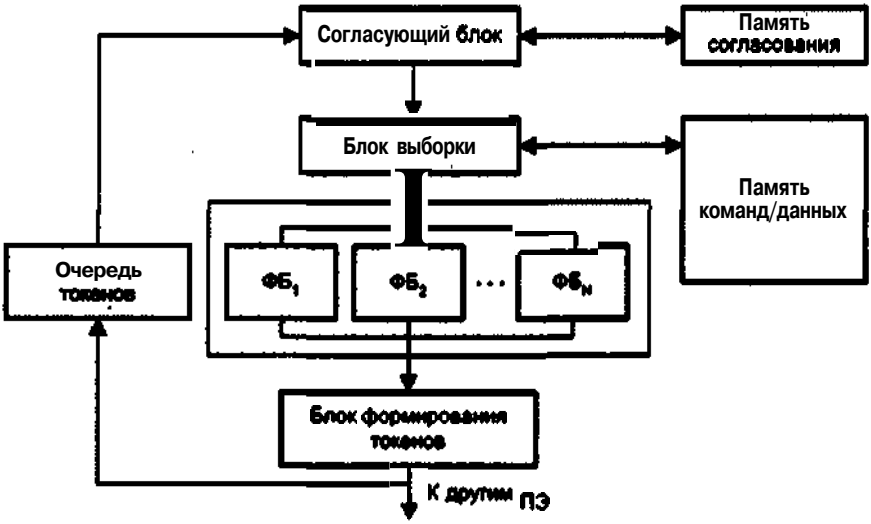


Рис. 15.11. Структура процессорного элемента типовой потоковой системы с помеченными токенами

Чтобы учесть возможные ситуации — циклы, элементы массивов, функции и рекурсии, — тег каждого токена должен включать в себя три поля: Уровень итерации • Имя функции • Индекс.

В каждом поле может содержаться число, начиная с нуля. Поле «Уровень итерации» хранит порядковый номер текущей итерации цикла; поле «Имя функции» идентифицирует вызов функции; поле «Индекс» указывает на определенный элемент массива. Первые два поля могут быть объединены в одно.

Индивидуальные поля тега трактуются как числовые значения, пересылаемые от одной вершины к другой. Минимальным требованием к динамической потоковой ВС является наличие операций, позволяющих извлечь значение, содержащееся в каждом поле, или установить в любом поле иное значение. Например, операция -«Прочитать поле токена»- берет токен и формирует из него другой токен, где определенное поле заполнено содержимым аналогичного поля из входного токена. Операция «Установить поле токена»- формирует выходной токен, совпадающий с входным, за исключением указанного поля, куда заносится значение, заданное в данной операции в виде литерала. Применительно к полю «Уровень итерации» могут быть предусмотрены операции инкремента и декремента.

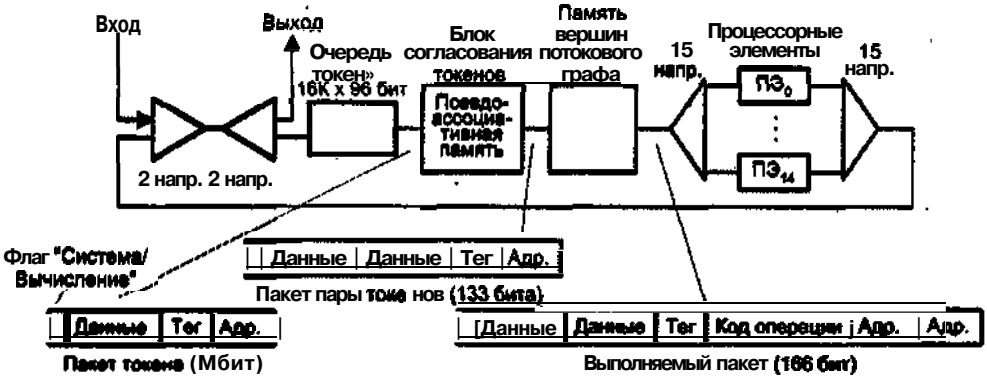


Рис. 15.12. Архитектура манчестерской потоковой вычислительной системы

На рис. 15.12 приведена структура динамической потоковой ВС с архитектурой окрашенных токенов, созданной в Манчестерском университете Гурдом и Ватсоном в 1980 году [114]. В этой ВС используется конвейерная кольцевая архитектура, и вычислительные функции отделены от функций согласования токенов и тегов. Каждый процессор обрабатывает пакеты длиной в 166 бит, содержащие два численных операнда, тег, код операции и один или два адреса следующей команды или команд. С каждым пакетом ассоциирован флаг «Система/Вычисление»-, позволяющий отличить выполняемый пакет (пакет, который должен быть обработан процессором) и пакет, несущий системное сообщение, такое, например,.как «Загрузить команду в память». Численными операндами могут быть 32-разрядные целые числа, либо числа в формате с плавающей запятой.

Обработав выполняемый пакет, процессор генерирует один или два пакета (токена) результата. Пакет результата состоит из 96 битов и содержит один численный операнд, тег и адрес (адреса) пункта назначения результата. Каждый пакет

результата поступает на ключ, обеспечивающий либо ввод данных от внешнего источника, либо вывод на внешний объект (периферийное устройство). Учитывая, что токен результата должен быть передан на другую вершину потокового графа, он направляется в очередь токенов, а оттуда — в блок согласования токенов. Здесь производится поиск других токенов, необходимых для инициирования вершины (если для этого нужны два токена). Поиск ведется аппаратно, посредством сравнения пункта назначения и тега входного токена с аналогичными параметрами всех хранящихся в памяти токенов. Если совпадение произошло, из пары токенов формируется пакет, в противном случае входной токен запоминается в блоке согласования токенов до момента, когда поступит согласующийся с ним токен. Пары токенов и токены с одним операндом передаются в память программ, содержащую узловые команды, и формируется полный выполняемый пакет. Выполняемые пакеты по возможности направляются в свободный процессор в массиве из 15 процессоров!

В качестве других примеров динамических потоковых вычислительных систем следует упомянуть: SIGMA-1 [123,124], PATTSY Processor Array Tagged-Token System [171], NTT's Dataflow Processor Array System [206], DDDP Distributed Data Driven Processor [143], SDFA Stateless Data-Flow Architecture [198].

Основное преимущество динамических потоковых систем — это более высокая производительность, достигаемая за счет допуска присутствия на дуге множества токенов. При этом, однако, основной проблемой становится эффективная реализация блока, который собирает токены с совпадающим цветом (токены с одинаковыми тегами). В плане производительности этой цели наилучшим образом отвечает ассоциативная память. К сожалению, такое решение является слишком дорогостоящим, поскольку число токенов, ожидающих совпадения тегов, как правило, достаточно велико. По этой причине в большинстве вычислительных систем вместо ассоциативных запоминающих устройств (ЗУ) используются обычные адресные ЗУ. В частности, сравнение тегов в рассмотренной манчестерской системе производится с привлечением хэширования, что несколько снижает быстродействие.

В последнее время все более популярной становится другая организация динамической потоковой ВС, позволяющая освободиться от ассоциативной памяти и известная как архитектура с явно адресуемыми токенами.

Архитектура потоковых систем с явно адресуемыми токенами

Значительным шагом в архитектуре потоковых ВС стало изобретение механизма *явной адресации токенов* (explicit token-store), имеющего и другое название - *непосредственное согласование* (direct matching). В основе этого механизма лежит то, что все токены в одной и той же итерации цикла и в одном и том же вхождении в реентерабельную процедуру имеют идентичный тег (цвет). При инициализации очередной итерации цикла или очередном обращении к процедуре формируется так называемый кадр токенов, содержащий токены, относящиеся к данной итерации или данному обращению, то есть с одинаковыми тегами, Использование конкретных ячеек внутри кадра задается на этапе компиляции. Каждому кадру вы-

деляется отдельная область в специальной памяти кадров (frame memory), причем раздача памяти под каждый кадр происходит уже на этапе выполнения программы.

Б схеме с явной адресацией токенов любое вычисление полностью описывается *указателем команды,(IP,* Instruction Pointer) *и указателем кадра* (FP, Frame Pointer). Этот кортеж <FP, IP> входит в тег токена, а сам токен выглядит следующим образом: Значение • FP.IP.

Команды, реализующие потоковый граф, хранятся в памяти команд и имеют формат: Код операции • Индекс в памяти кадров • Адресат.

Здесь «Индекс в памяти кадров» определяет положение ячейки с нужным токеном внутри кадра, то есть какое:число нужно добавить к FP, чтобы получить адрес интересующего токена. Поле -«Адресат» указывает на местоположение команды, которой должен быть передан результат обработки данного токена. Адрес в этом поле также задан в виде смещения — числа, которое следует прибавить к текущему значению IP, чтобы получить исполнительный адрес команды назначения в памяти команд. Если потребителей токена несколько, в поле «Адресат» заносится несколько значений смещения. Простой пример кодирования потокового графа и токенов на его дугах показан на рис. 15.13.

Каждому слову в памяти кадров придан *бит наличия,* единичное значение которого удостоверяет, что в ячейке находится токен, ждущий согласования, то есть что одно из искомых значений операндов уже имеется. Как и в архитектуре с окра-

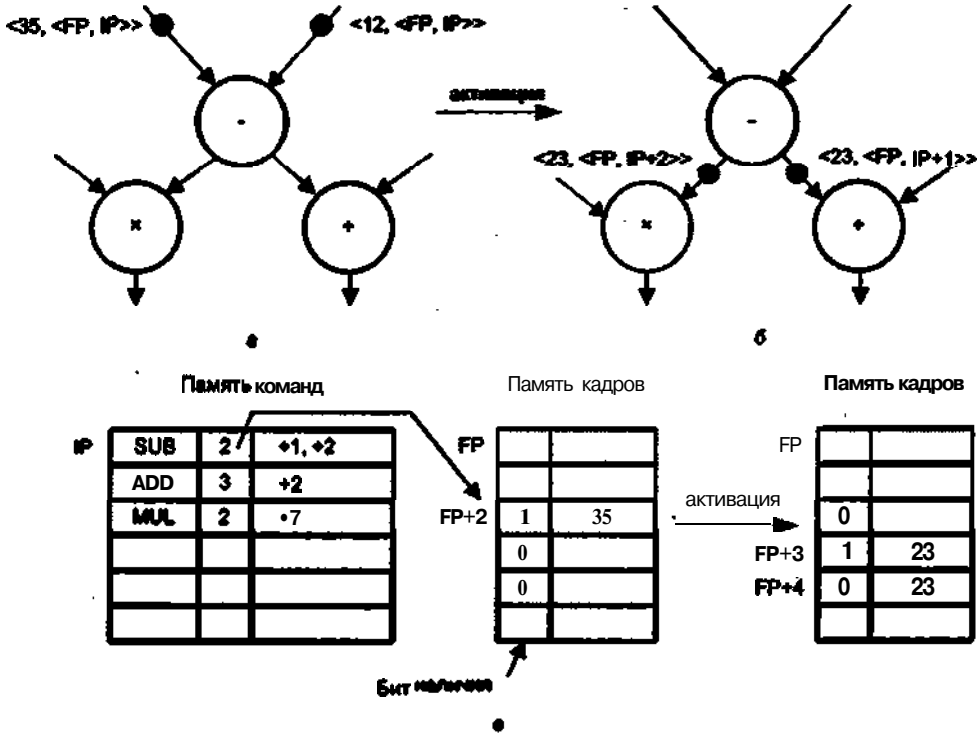


Рис 15.13. Кодирование в архитектуре с явной адресацией токенов: *а* — активизация вершины вычитания; *б* — активизация вершин умножения и деления; *в* — кодирование потокового графа

шенными токенами, определено, что вершины могут иметь максимум две входные дуга. Когда на входную дугу вершины поступает токен <vl, <FP, IP», в ячейке памяти кадров с адресом FP + (IP.I) проверяется бит наличия (здесь IP.I означает содержимое поля I в команде, хранящейся по адресу, указанному в IP). Если бит наличия сброшен (ни один из пары токенов еще не поступал), поле значения пришедшего токена (vl) заносится в анализируемую ячейку памяти кадров, а бит наличия в этой ячейке устанавливается в единицу, фиксируя факт, что первый токен из пары уже доступен:

```
(FP + (IP.I).значение := vl
(FP + (IP.I).наличие := 1
```

Этот случай отражен на рис. 15.13, *а,* когда на вершину SUB по левой входной дуге поступил токен <*35,* <FP, IP».

Если токен <v2,<FP, IP>> приходит на узел, для которого уже хранится значение vl, команда, представляющая данную вершину, может быть активирована и выполнена с операндами vl и v2. В этот момент значение vl извлекается из памяти кадров, бит наличия сбрасывается, и на функциональный блок, предназначенный для выполнения операции, передается пакет команды <vl, v2, FP, IP, IP.OP, IP.0>, содержащий операнды (vl и v2), код операции (IP.OP) и адресат ее результата (IP.D). Входящие в этот пакет значения FP и IP нужны, чтобы вместе с IP.D вычислить исполнительный адрес адресата. После выполнения операции функциональный

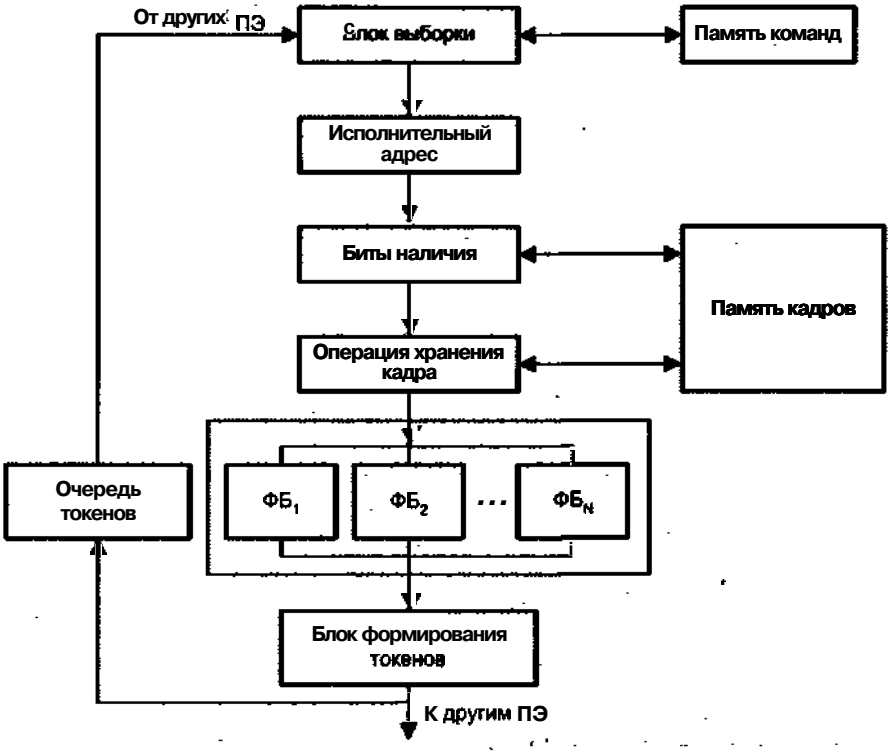


Рис. 15.14. Структура процессорного элемента типовой потоковой системы с явной адресацией токенов

блок пересылает результат в блок формирования токенов. Рисунок 15.13, б демонстрирует ситуацию, когда токен уже пришел и на второй вход вершины SUB. Операция становится активируемой, и после ее выполнения результат передается на вершины A DD и MUL, которые ожидают входных токенов в ячейках FP+3 и FP+4 соответственно.

Типовая архитектура системы с явной адресацией токенов показана на рис. 15.14. Отметим, что функция согласования то'кенов стала достаточно короткой операцией, что позволяет внедрить ее в виде нескольких ступеней процессорного конвейера.

Макропотоковые вычислительные системы

Рассмотренный ранее механизм обработки с управлением от потока данных функционирует на уровне команд и его *относят к потоковой обработке низкого уровня* (fine-grain dataflow). Данному подходу сопутствуют большие издержки при пересылке операндов. Для уменьшения коммуникационных издержек необходимо применять потоковую обработку на процедурном уровне, так называемую *укрупненную потоковую* или *макропотоковую* обработку (multithreading). Буквальный перевод английского термина означает потоковую обработку множества нитей.

Макропотоковая модель совмещает локальность программы, характерную для фон-неймановской модели, с толерантностью к задержкам на переключение задач, свойственной потоковой архитектуре. Это достигается за счет того, что вершина графа представляет собой не одну команду, а последовательность из нескольких команд, называемых *нитью* (thread). По этой причине макропотоковую организацию *часто* и *меню*т *крупнозернистой потоковой обработкой* (coarse-grained dataflow). Макропотоковая обработка сводится к потоковому выполнению нитей, в то время как внутри отдельной нити характер выполнения фон-неймановский. Порядок обработки нитей меняется динамически в процессе вычислений, а последовательность команд в пределах нити определена при компиляции статически. Структура макропотоковой ВС представлена на рис. 15.15.

Существенное отличие макропотоковой системы от обычной потоковой ВС состоит в организации внутреннего управляющего конвейера, где последовательность выполнения команд задается счетчиком команд, как в фон-неймановских машинах. Иными словами, этот конвейер идентичен обычному конвейеру команд.

Вернемся к иллюстрации возможных вычислительных моделей (см. рис.15.1). В макропотоковой архитектуре (см. рис. 15.1, *в)* каждый узел графа представляет команду, а каждая закрашенная область — одну из нитей. Если команда приостанавливается, останавливается и соответствующая нить, в то время как выполнение других нитей может продолжаться.

Существуют две формы макропотоковой обработки: *без блокирования и с блокированием,* В модели без блокирования выполнение нити не может быть начато, пока не получены все необходимые данные. Будучи запущенной, нить выполняется до конца без приостановки. В варианте с блокированием запуск нити может быть произведен до получения всех операндов. Когда требуется отсутствующий операнд, нить приостанавливается (блокируется), а возобновление выполнения откладывается на некоторое время. Процессор запоминает всю необходимую информацию о состоянии и загружает на выполнение другую готовую нить. Модель с блокированием обеспечивает более мягкий подход к формированию нитей (часто это выражается в возможности использования более длинных нитей) за счет дополнительной аппаратуры для хранения блокированных нитей.

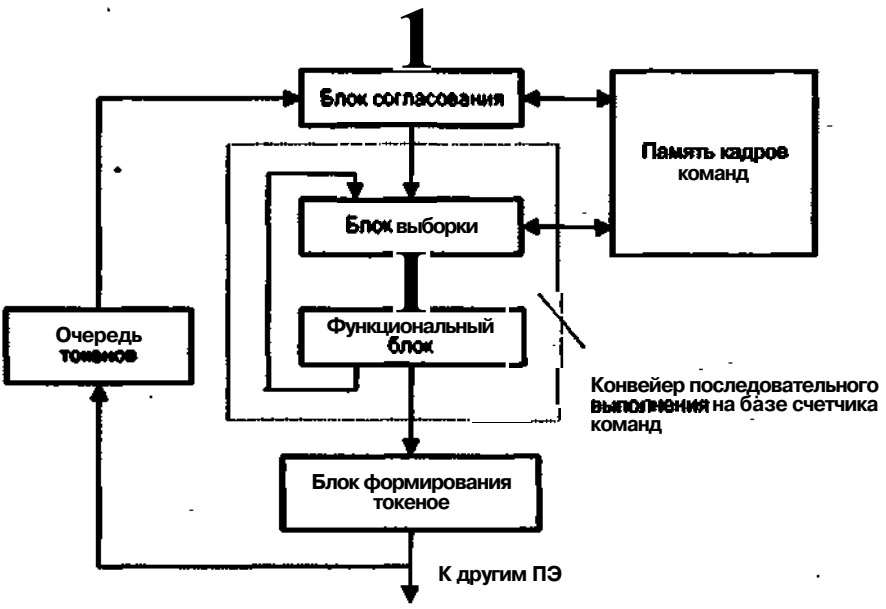


Рис. 15.15. Структура процессорного элемента типовой макропотоковой системы

Возможна также и потоковая обработка переменного уровня, когда узлы соответствуют как простым операциям, так и сложным последовательным процедурам. Последний случай иногда называют *комбинированной обработкой с потоками данных и потоками управления* (combined dataflow/control flow).

Гиперпотоковая обработка

В *основе гиперпотоковой технологии* (hyperthreading), разработанной фирмой Intel и впервые реализованной в микропроцессоре Pentium 4, лежит то, что современные процессоры в большинстве своем являются суперскалярными и многоконвейерными, то есть выполнение команд в них идет параллельно, по этапам и на нескольких конвейерах сразу. Гиперпотоковая обработка призвана раскрыть этот потенциал таким образом, чтобы функциональные блоки процессора были бы максимально загружены. Поставленная цель достигается за счет сочетания соответствующих аппаратных и программных средств.

Выполняемая программа разбивается на два параллельных потока (threads). Задача компилятора (на стадии подготовки программы) и операционной системы (на этапе выполнения программы) заключается в формировании таких последовательностей независимых команд, которые процессор мог бы обрабатывать параллельно, по возможности заполняя функциональные блоки, не занятые одним из потоков, подходящими командами из другого, независимого потока.

Операционная система, поддерживающая гиперпотоковую технологию, воспринимает физический суперскалярный процессор как два логических процессора и организует поступление на эти два процессора двух независимых потоков команд.

Процессор с поддержкой технологии hyperthreading эмулирует работу двух одинаковых логических процессоров, принимая команды, направленные для каж-

дого из них. Это не означает, что в процессоре имеются два вычислительных ядра — оба логических процессора конкурируют за ресурсы единственного вычислительного ядра,' Следствием конкуренции является более эффективная загрузка всех ресурсов процессора,

В процессе вычислений физический процессор рассматривает оба потока команд и по очереди запускает на выполнение команды то из одного, то из другого, или сразу их двух, если есть свободные вычислительные ресурсы. Ни один из потоков не считается приоритетным. При остановке одного из потоков (в ожидании какого-либо события или в результате зацикливания) процессор полностью переключается на второй поток. Возможность чередования команд из разных потоков составляет принципиальное отличие между гиперпотоковой и макропотоковой обработкой.

Наличие только одного вычислительного ядра не позволяет достичь удвоенной производительности, однако за счет большей отдачи от всех внутренних ресурсов общая скорость вычислений существенно возрастает. Это особенно ощущается, когда потоки содержат команды разных типов, тогда замедление обработки в одном из них компенсируется большим объемом работ, выполненных в другом потоке.

Следует учитывать, что эффективность технологии hyperthreading зависит от работы операционной системы, поскольку разделение команд на потоки осуществляет именно она.

Для иллюстрации рассмотрим некоторые особенности реализации гиперпотоковой технологии в процессоре Pentium 4 Хеоп. Процессор способен параллельно обрабатывать два потока в двух логических процессорах. Чтобы выглядеть для операционной системы и пользователя как два логических процессора, физический процессор должен поддерживать информацию одновременно для двух отдельных и независимых потоков, распределяя между ними свои ресурсы. В зависимости от вида ресурса применяются три подхода: дублирование, разделение и совместное использование.

Дублированные ресурсы. Для поддержания двух полностью независимых контекстов на каждом из логических процессоров некоторые ресурсы процессора необходимо дублировать. Прежде всего, это относится к счетчику команд (IP, Instruction Pointer), позволяющему каждому из логических процессоров отслеживать адрес очередной команды потока. Для параллельного выполнения нескольких процессов необходимо столько IP, сколько потоков команд необходимо отслеживать одновременно. Иными словами, у каждого логического процессора должен быть свой счетчик команд. В процессоре Хеоп максимальное количество потоков команд равно двум и поэтому требуется два счетчика команд. Кроме того, в процессоре имеются две таблицы распределения регистров (RAT, Register Allocation Table), каждая из которых обеспечивает отображение восьми регистров общего назначения (РОН) и восьми регистров с плавающей запятой (РПЗ), относящихся к одному логическому процессору, на совместно используемый регистровый файл из 128 РОН и 128 РПЗ. Таким образом, RAT - это дублированный ресурс, управляющий совместно используемым ресурсом (регистровым файлом).

Разделенные ресурсы- В качестве одного из видов разделенных ресурсов в Хеоп выступают очереди (буферная память, организованная по принципу FIFO), рас-

положенные между основными ступенями конвейера. Применяемое здесь разделение ресурсов можно условно назвать статическим: каждая буферная память (очередь) разбивается пополам, и за каждым логическим процессором закрепляется своя половина очереди.

Применительно к другому виду очередей — очередям диспетчеризации команд (их в процессоре три) — можно говорить о динамическом разделении. Вместо того чтобы из предусмотренных в каждой очереди двенадцати входов фиксировано назначить входы 0-5 логическому процессору (ЛП) 0, а входы 6-11 — логическому процессору 1, каждому ЛП разрешается использовать любые входы очереди, лишь бы их общее число не превысило шести.

С позиций логического процессора и потока между статическим и динамическим разделением нет никакой разницы — в обоих случаях каждому ЛП выделяется своя половина ресурса. Различие становится существенным, если в качестве отправной точки взять физический процессор. Отсутствие привязки потоков к конкретным входам очереди позволяет не принимать во внимание, что имеются два потока, и расценивать обе половины как единую очередь. Очередь диспетчеризации команд просто просматривает каждую команду в общей очереди, оценивает зависимости между командами, проверяет доступность ресурсов, необходимых для выполнения команды, и планирует команду к исполнению. Таким образом, выдача команд на исполнение не зависит от того, какому потоку они принадлежат. Динамическое разделение очередей диспетчеризации команд предотвращает монополизацию очередей каким-либо одним из логических процессоров.

Завершая обсуждение разделяемых ресурсов, отметим, что если процессор Хеоп обрабатывает только один поток, то для обеспечения максимальной производительности этому потоку предоставляются все ресурсы процессора В динамически разделяемых очередях снимаются ограничения на количество входов, доступных одному потоку, а в статических разделяемых очередях отменяется их разбиение на две половины.

Совместно используемые ресурсы. Этот вид ресурсов в гиперпотоковой технологии считается определяющим. Чем больше ресурсов могут совместно использовать логические процессоры, тем большую вычислительную мощность можно «снять» с единицы площади кристалла процессора. Первую группу общих ресурсов образуют функциональные (исполнительные) блоки: целочисленные операционные устройства, блоки операций с плавающей запятой и блоки обращения (чтения/записи) к памяти. Эти ресурсы "не знают", из какого ЛП поступила команда. То же самое можно сказать и о регистровом файле — втором виде совместно используемых ресурсов.

Сила гиперпотоковой технологии — общие ресурсы — одновременно является и ее слабостью. Проблема возникает, когда один поток монополизирует ключевой ресурс (такой, например, как блок операций с плавающей запятой), чем блокирует другой поток, вызывая его остановку. Задача предотвращения таких ситуаций возлагается на компилятор и операционную систему, которые должны образовать потоки, состоящие из команд с максимально различающимися требованиями к совместно используемым ресурсам. Так, один поток может содержать команды, нуждающиеся главным образом в блоке для операций с плавающей запятой, а дру-

гой - состоять преимущественно из команд целочисленной арифметики и обращения к памяти,

В заключение необходимо остановиться на третьем виде общих ресурсов - кэш-памяти. Процессор Хеоп предполагает работу с кэш-памятью трех уровней (LI, L2 и L3) и так называемой кэш-памятью трассировки. Оба логических процессора совместно используют одну и ту же кэш-память и хранящиеся в ней данные. Если поток, обрабатываемый логическим процессором 0, хочет прочитать некоторые данные, кэшированные логическим процессором 1, он может взять их из общего кэша. Из-за того, что в гиперпотоковом процессоре одну и ту же кэш-память используют сразу два логических процессора, вероятность конфликтов и, следовательно, вероятность снижения производительности возрастает.

Любой вид кэш-памяти одинаково трактует все обращения для чтения или записи, вне зависимости от того, какой из логических процессоров данное обращение производит. Это позволяет любому потоку монополизировать любой из кэшей, причем никакой защитой от монополизации, как это имеет место в случае очередей диспетчеризации команд, процессор не обладает. Иными словами, физический процессор не в состоянии заставить логические процессоры сотрудничать при их обращении к кэшам.

В целом, среди совместно используемых ресурсов в технологии hyperthreading кэш-память оказывается наиболее критичным местом, и конфликты за обладание этим ресурсом сказываются на общей производительности процессора наиболее ощутимо.

По оценке Intel, прирост скорости вычислений в некоторых случаях может достигать 25-35%. В приложениях, ориентированных на многозадачность, программы ускоряются на 15-20%. Возможны, однако, ситуации, когда прирост в быстродействии может быть незаметен и даже быть отрицательным. Таким образом, эффективность технологии находится в прямой зависимости от характера реализуемого программного приложения. Максимальная отдача достигается при работе серверных приложений за счет разнообразия процессорных операций.

В настоящий момент аппаратная поддержка технологии заложена в микропроцессоры Pentium 4, причем, по информации Intel, в процессоре Pentium 4 Хеоп это потребовало 5% дополнительной площади на кристалле. Программная поддержка технологии предусмотрена в операционных системах Windows 2000, Windows XP и Windows .NET Server (в предшествующих ОС Windows такая возможность отсутствует).

Вычислительные системы с управлением вычислениями по запросу

В системах с управлением от потока данных каждая команда, для которой имеются все необходимые операнды, немедленно выполняется. Однако для получения окончательного результата многие из этих вычислений оказываются ненужными. Отсюда прагматичным представляется иной подход, когда вычисления инициируются не по готовности данных, а на основе запроса на данные. Такая организация *вычислительного процесса носит название управления вычислениями по запросу*

(demand-driven control). В ее основе, как и в потоковой модели (data-driven control), лежит представление вычислительного процесса в виде графа. В потоковой модели узлы вверху графа запускаются раньше, чем нижние. Это — нисходящая обрабо́тка. Механизм управления по запросу состоит в обработке вершин потокового графа снизу вверх путем разрешения запуска узла, лишь когда требуется его результат. Данный процесс получил *название редукции графа,* а ВС, оперирующая в режиме снизу вверх (см, рис. 15.1, г), называется *редукционной вычислительной системой.*

Математическую основу редукционных ВС составляют *лямбда-исчисления* [2, 57,202], а для написания программ под такие системы нужны так называемые функциональные языки программирования (FP, Haskell и др.). На функциональном языке все программы представляются в виде выражений, а процесс выполнения программы заключается в определении значений последних (это называется *оценкой выражения).* Оценка выражения производится посредством повторения операции выбора и упрощения тех частей выражения, которые можно свести от сложного к простому (такая часть выражения *называется редексом,* причем сам редекс также является отдельным выражением). Операция упрощения называется *редукцией.* Процесс редукции завершается, когда преобразованное редукцией выражение больше не содержит редекса. Выражение, не содержащее редекса, называется *нормальной формой.*

В редукционной ВС вычисления производятся по запросу на результат операции. Предположим, что вычисляется выражение a = (b +1) x c — .В случае потоковых моделей процесс начинается с самых внутренних операций, а именно с параллельного вычисления (b + 1) и *d/c.* Затем выполняется операция умножения *(b* + 1) x c и, наконец, самая внешняя операция — вычитание. Такой род вычислений часто *называют энергичными вычислениями* (eager evaluation).

При вычислениях, управляемых запросами, все начинается с запроса на результат *a,* который включает в себя запрос на вычисление выражений *(Ъ* + 1) x c и *d/c,* а те, в свою очередь, формируют запрос на вычисление b + 1, то есть на операцию самого внутреннего уровня. Результат возвращается в порядке, обратном поступлению запросов. Отсюда *название ленивые вычисления* (lazy evaluation), поскольку операции выполняются только тогда, когда их результат требуется другой команде. Редукционные вычисления, естественно, согласуются с концепцией функционального программирования, упрощающей распараллеливание программ.

На рис. 15.16 показан процесс вычисления с помощью редукционной ВС зна*чентвырзжешя a =b - c(Ъ =d + e,c=fxg) для d* =1,e = 3,f=5,g=7.Программа редукции состоит из распознавания редексов с последующей заменой их вычисленными значениями. Таким образом, вся программа в конечном итоге редуцируется до результата.

Известны два типа моделей редукционных систем: строчная и графовая, отличающиеся тем, что именно передается в функцию - скопированные значения данных или же только указатели, указывающие на места хранения данных.

В *строчной редукционной модели* каждый запросивший узел получает отдельную копию выражения для собственной оценки. Длинное строковое выражение

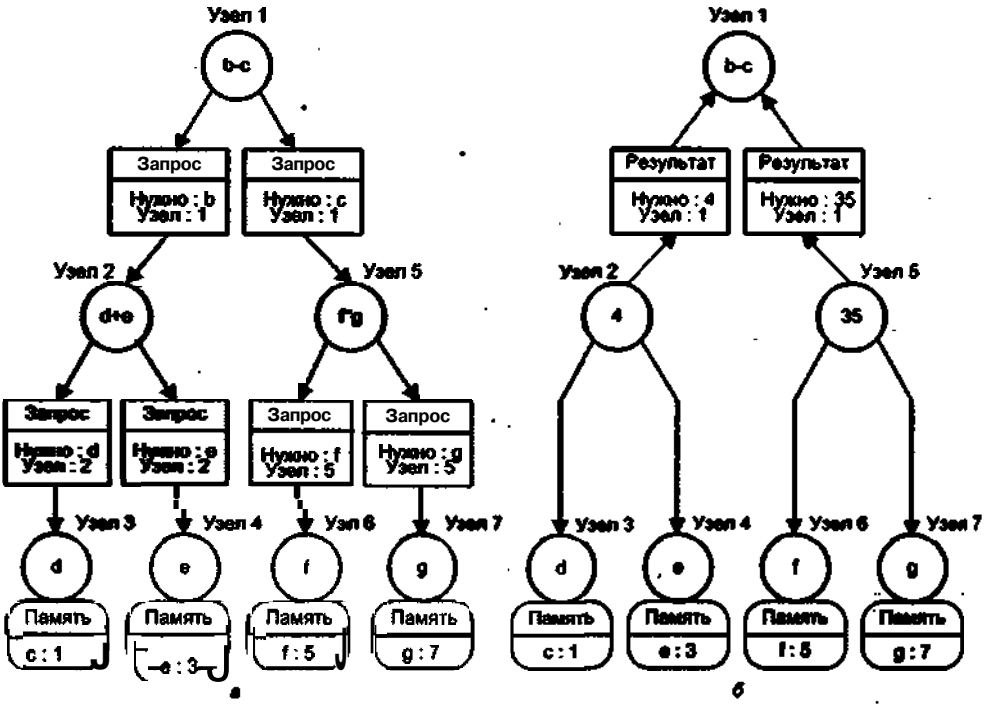


Рис. 15.16. Пример вычисления выражения на редукционной вычислительной системе: *а* — исходное положение; *б* — после первого шага редукции

рекурсивным образом сокращается (редуцируется) до единственного значения. -Каждый шаг редукции содержит оператор, сопровождаемый ссылкой на требуемые входные операнды. Оператор приостанавливается, пока оцениваются входные параметры.

На рис. 15.17 показан процесс вычислений с помощью строчной редукции. Если требуется значение $a=(b + c) \times (b - c)$ (рис. 15.17, я), то копируется граф программы, определяющий вычисление *а* (рис. 15.17,*б*}. При этом запускается операция умножения. Поскольку это вычисление невозможно без предварительного расчета двух параметров, то запускаются вычисления «+» и «-», в результате чего образуется редуцированный граф (с ветвями «6» и «2»), показанный на рис, 15.17, *в*. Результат получается путем дальнейшей редукции (рис. 15.17, г),

В *графовой редукционной модели выражение* представлено как ориентированный граф. Граф сокращается по результатам оценки ветвей и подграфов. В зависимости от запросов возможно параллельное оценивание и редукция различных частей графа или подграфов. Запросившему узлу, который управляет всеми ссылками на граф, возвращается указатель на результат редукции. «Обход» графа и изменение ссылок продолжаются, пока не будет получено значение результата, копия которого возвращается запросившей команде.

Рисунок 15.18 иллюстрирует пример вычислений с помощью графовой редукции. В этой модели, когда требуется найти значение *а,* определение вычисления *а* не копируется, а передается указатель определяющей программы. При достиже-

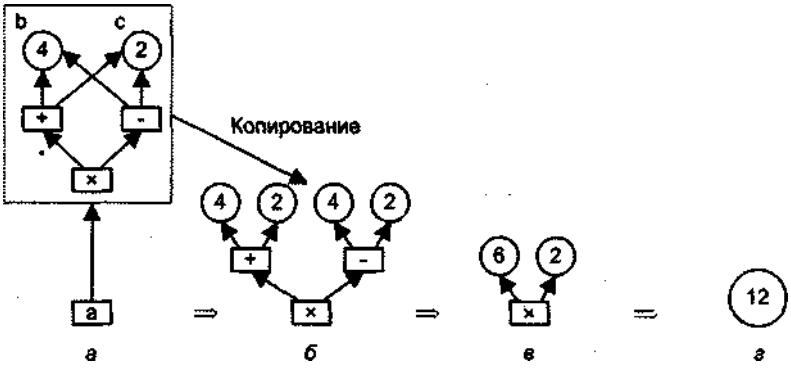


Рис. 15.17. Процесс вычислений в модели со строчной редукцией: *в* — исходный граф; *б, в* — последовательно редуцированные графы; г— результат редукции

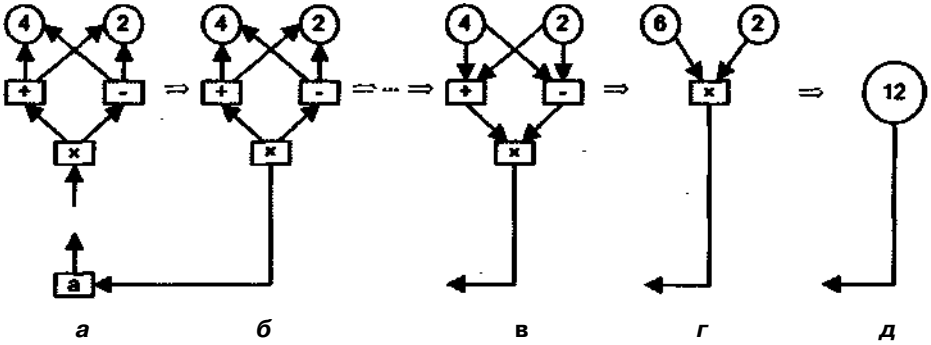


Рис. 15.18. Процесс вычислений в модели с графовой редукцией: *а* - исходный граф; б, в, г— последовательная редукция со сменой направления указателей; Л — результат редукции

нии узла «Ч» направление переданного указателя меняется на противоположное, чтобы запомнить место, из которого будет выдаваться результат вычисления (рисунок 15.18, *б)*. Далее путем повторения операции смены направления указателя (текущего указателя) на обратное получается граф, показанный на рис. 15.18, *в*. Теперь операции *+* и «-» можно выполнять, граф редуцируется сначала до изображенного на рис. 15.18, г, а затем — на рис. 15.18,5.

Контрольные вопросы

1. Перечислите и охарактеризуйте возможные механизмы управления вычислительным процессом.
2. В чем состоит идея управления от потока данных?
3. Какие элементарные операторы могут быть взяты в качестве вершин потокового графа?
4. Каким образом осуществляется передача данных между узлами потокового графа?

5. В чем состоит принципиальное различие между статической и динамической потоковой архитектурами?
6. Выполнение какого условия, кроме наличия входных данных, требуется для активации операции в статической потоковой ВС?
7. Опишите структуру пакетов действий и пакетов результата в статической потоковой ВС и поясните назначение полей этих пакетов.
8. Какой смысл вкладывается в понятие "окрашенный токен"?
9. Сохраняется ли в потоковых ВС принцип локальности по обращению, свойственный вычислительным системам фон-неймановского типа?
10. Определите понятие thread применительно к макропотоковой обработке.
11. В чем заключаются преимущества макропотоковой обработки над обычной потоковой?
12. Каким образом и при каких условиях гиперпотоковая обработка способствует повышению производительности процессора?
13. Почему вычислительные системы с управлением по запросу называют редукционными?
14. Какой математический аппарат лежит в основе редукционных ВС? Поясните основные положения этого аппарата.
15. Поясните различия между строковой и графовой моделями редукции.

Заключение

Любую работу трудно начинать и еще труднее заканчивать, но приходится...

Наши поздравления уважаемому читателю - надеемся, что вы оказались на этой странице не в силу природного любопытства и нетерпеливости, а в результате изучения всего материала учебника.

Теперь вы вооружены и опасны ©. Вооружены базовыми знаниями в данной предметной области, а опасны для дилетантов и «незнаек», то есть людей несведущих. И, конечно, вы открыты новым знаниям, тому, что у нас впереди. Это очень важно, ведь темпы развития в этой области знаний предельно высоки. Специалист по вычислительным машинам и системам должен быть готов к обучению на протяжении всей профессиональной жизни: поезд новых компьютерных решений движется чрезвычайно стремительно, только успевай впрыгивать на его подножку!

Мы намеренно не употребляли слово -«электронные» применительно к вычислительным машинам. Перефразируя известное высказывание, электроника — колыбель вычислительных машин, но нельзя же вечно жить в колыбели! Двадцать первый век принесет массу сюрпризов в области элементной базы ВМ и ВС, а вместе с ее изменениями переменится архитектура и организация вычислительных средств. Вот краткий список тех новаций, которые уже стучатся в дверь: голографическая, твердотельная и протонная память; схемы на базе молекулярных ключей; оптические, квантовые и нанокомпьютеры; электронная цифровая бумага; пластмассовые дисплеи; нейроинформатика, биоинформатика... И это еще далеко не все. Словом, дорога в компьютерный космос открыта, а информационная революция только начинается... Будьте готовы к переменам, и все у вас получится :).

Впереди длинный и интересный путь познаний. Удачи вам, уважаемый читатель, на этом пути!

Список литературы

1. *Авен О. И, Гурин Н. Н,, Коган А. Я,* Оценка качества и оптимизации вычислительных систем. М.: Наука, 1982.464 с.
2. *Амамия М., Танака Ю.* Архитектура ЭВМ и искусственный интеллект. М.: Мир, 1993.400 с.
3. *Баранов С. И., Баркалов А. А.* Микропрограммирование: принципы, методы, применения. Зарубежная радиоэлектроника, 1984, № 5. С. 3-29.
4. *Воеводин В. В., Воеводин Вл. В.* Параллельные вычисления. СПб.: БХВ-Петербург, 2002. 608 с.
5. *Воеводин Вл. В., Капитонова А. П.* Методы описания и классификации вычислительных систем. М.: Изд. МГУ, 1994.103 с,
6. *Волков Д.* Как оценить рабочую станцию. Открытые системы, 1994, №2. С. 44-48.
7. *Волков А. А.* Тесты TPC. СУБД, 1995, № 2. С. 70-78.
8. *Дубова Н.* Суперкомпьютеры nCube. Открытые системы, 1995, № 2. С. 42-47.
9. *Злотник Е. М.* Секционированные микропроцессоры. Минск: Наука и техника, 1984.191 с.
10. *Каган Б. М.* Электронные вычислительные машины и системы. М.: Энергоатомиздат,1991.592с.
11. *Клейнрок Л.* Вычислительные системы с очередями. М.: Мир, 1979. 600 с.
12. *Колосов В. Г., Мелехин В. Ф.* Проектирование узлов и систем автоматики и вычислительной техники. Л.: Энергоатомиздат, 1983.256 с.
13. Котов В.Е.СетиПетри.М.:Наука,1984,160с.
14. *Крайзмер Л. П., Бородаев Д А., Гутенмахер Л. И., Кузьмин Б. Н., Смелянский И. Л.* Ассоциативные запоминающие устройства. Л.: Энергия, 1967.
15. *Кузьминский М.* Между строк таблиц Unpack. Computerworld, 1998, № 13.
16. *Кургаев А. Ф Писарский А. В.* Об оценке эффективности системы команд ЭВМ. УСИМ, 1981, № 1. С. 40-44.

17. *Кургаев А. Ф., Полтин А. В., Писарский А. В., Юсифов С. И.* О выборе базового набора операторов // В кн.: Разработка средств кибернетической техники. Киев: ИК АН УССР, 1982. С. 3-8.

18. *Ладенко И. С.* Имитационные системы (методология исследований и проектирования). СО АН СССР. Институт экономики и организации промышленного производства. Новосибирск: Наука. Сибирское отделение, 1981.300 с.

19. *Ларионов А. М.* Вычислительные комплексы, системы и сети. М.: Энергоатомиздат, 1987.287 с.

20. *Линский В. С.* О выборе рационального количества адресов цифровой вычислительной машины // В кн.: Вопросы теории математических машин, 1958, вып. 1. С. 181-191.

21. *Майоров С. А., Новиков Г И.* Структура электронных вычислительных машин. Л.: Машиностроение, 1979.384 с.

22. *Новиков Г. И., Павлов В. П.* Способ определения оптимального набора микроопераций и логических условий. УСИМ, 1979, № 4. С. 90-95.

23. *Опадчий Ю. Ф., Глудкин О. П, Гуров А, И.* Аналоговая и цифровая электроника. М.: Горячая линия - Телеком, 2002.768 с.

24. *Орлов С. А.* Управляющие ЭВМ. М.: Изд-во МО, 1981.241 с.

25. *Орлов С. А.* Организация и проектирование цифровых управляющих микроЭВМ и микроВС. М.: Изд-во МО, 1985.475 с.

26. *Орлов С. А.* Основы организации микропроцессорных средств автоматизированных систем. Учебное пособие. М.: Изд-во МО, 1986.207 с.

27. Основы современных компьютерных технологий // Под ред. А. Д. Хомоненко. СПб: КОРОНА принт, 1998.448 с.

28. *Палаши А. В., Иванов В. А., Кургаев А. Ф„ Денисенко В. П.* МиниЭВМ: Принципы построения и проектирования. Киев: Наукова думка, 1975.200 с,

29. *Панфилов И. В., Половко А. М,* Вычислительные системы. М.: Советское радио, 1990.304 с.

30. *Паулин Г.* Малый толковый словарь по вычислительной технике // Пер. с нем. М.: Энергия, 1975.

31. *ПитерсонДж.* Теория сетей Петри и моделирования систем // Пер. с англ. М.: Мир, 1984.264 с.

32. Проектирование цифровых систем на комплектах микропрограммируемых БИС // Булгаков С. С, Мещеряков В. М„ Новоселов В. В., Шумилов Л. А. М.: Радио и связь, 1984.240 с.

33. *Пятибратов А. П., Гудыно Л. П., Кириченко А. А.* Вычислительные системы, сети и телекоммуникации. М.: Финансы и статистика, 1998.400 с.

34. *Смелянский Р. Л., Бахмуров А. Г., Гурьев Д.* Об одной вероятностной модели программ. Программирование, 1986, № 6.

35. *Смелянский Р. Л.* Методы анализа и оценки производительности вычислительных систем. М.: МГУ, 1990.

36. *Столлингс У.* Структурная организация и архитектура компьютерных систем, 5-е изд. // Пер. с англ. М.: Изд. дом «Вильяме», 2002.896 с.

37. Толковый словарь по вычислительным системам // Пер. с англ. М.: Машиностроение, 1990.568 с.

38. *ФеррариД.* Оценка производительности вычислительных систем //Пер. с англ. М.: Мир, 1981.576 с.

39. *Харкевич А А.* Борьба с помехами. М.: Госиздат физико-математической литературы, 1963. 276с.

40. *Хокни Р., Джессхоуп К.* Параллельные ЭВМ: Архитектура, программирование и алгоритмы. М.: Радио и связь. 1986.392 с.

41. *Цилъкер Б. Я., Макеев В. Я.* Архитектура вычислительных машин. Рига: TSI, 2000.213 с.

42. *Цилъкер Б. Я., Пятков В. П.* Архитектура вычислительных систем. Рига: TSI, 2001.249 с.

43. *Шнитман В. З.* Системы Exemplar SPP1200. Открытые системы, 1995, № 6.

44. *Agarwal, A., Bianchmi, R., Chatken, D.Johnson, K. L, Kranz>D., KubiatowiczJ., Lim, B-H., Mackenzie, K., Yeung, D.* «The MIT Alewife Machine: Architecture and Performance», Proceedings of the 22ii Annual Internationa] Symposium on Computer Architecture, Jun. 1995, pp. 2-13.

45. *Agerwala, T., Cocke,J.* «High Performance Reduced Instruction Set Processors»-, Technical Report RC12434 (#55845), Yorktown, New York: IBM Thomas J. Watson Research Centerjan. 1987.

46. *Almost, G. S„ Gottlieb, A.* «Highly Parallel Computing», 2nd Edition, Addison-Wesley, 1994.

47. *AltnetherJ.* «Error Detecting and Correcting Codes»-, Intel Application Note AP-46,1979.

48. *Amdahl G. M.* «Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities», Proceedings AFIPS Conference, Vol. 30 (Atlantic City, New Jersey, Apr. 18-20), AFIPS Press, Reston, Va, 1967, pp. 483-485.

49. *Andrews, M.* «Principles of Firmware Engineering in Microprogram Control. Silver Spring»-, MD, Computer Science Press, 1980.

50. *Andrews, W.* «Futurebus+ Spec Completed», Almost Computer Design, Feb. 1, 1990?, pp. 22-28.

51. *Archibald,J. A.* «The Cache Coherence Problems in Shared-Memory Multi-processors», Technical Report, University of Washington, Feb. 1987.

52. *Archibald, J. A.* «Cache Coherence Approach for Large Multiprocessor Systems», Proceedings of the 2nd International Conference on Supercomputing, ACM, New York, 1988, pp. 337-345.

53. *Arvind, Kathail, V.* «A Multiple Processor Dataflow Machine that Supports Generalized Procedures», Proceedings 8th ISCA, May 1981, pp. 291-302.

54. *Arvind, Nikhil, R. S.* «Executing a Program on the MIT Tagged-Token Dataflow Architecture», IEEE Transactions Computers, C-39, 1990, pp. 300-318.

55. *Arvind, Bic, L., Ungerer, T.* «Evolution of Dataflow Computers. Advanced Topics in Data-Flow Computing» (Gaudiot J-L., and Bic L., eds.), Prentice-Hall, 1991, pp. 3-33.

56. *Backus, J. W.* «Can Programming be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs», CACM, Vol.21, 1978, pp. 613-641.

57. *Barendregt, H. P.* «The Lambda Calculus - Its Syntax and Semantics», 2nd Edition, North-Holland, 1984.

58. *Barnes, G. H.. Brown, R.M., Kato, M.,Kuck, D.J., Slotnick, D. L.,Stokes, R. A.* «The ILLIAC IV Computer», IEEE Transactions Computers, Aug. 1968, pp. 746-757.

59. *Basu, A.* «Parallel Processing Systems: A Nomenclature Based on their Characteristics». Proceedings IEE (UK), №134, 1987, pp. 143-147.

60. *Batcher, K. E.* «Sorting Networks and their Applications», Proceedings SJCC, 1968, pp. 307-314.

61. *Baugh, C. R., Wooley, B. A.* «A Two's Complement Parallel Array Multiplication Algorithm», IEEE Transactions on Computers, C-22, Dec. 1973, pp. 1045-1047.

62. *Benes, V.E.* «Mathematical Theory of Connecting Networks and Telephone Traffic», Academic Press, 1965, 53 pp.

63. *Booth, A. D.* «A Signed Binary Multiplication Technique», Quart. J. Mech. Appl. Math. 4, part 2, 1951, pp. 236-240.

64. Branch Prediction Techniques, 1995, Available at http://www.umiacs.umd.edu/~hismail/818K/node7.html.

65. *Brewer, E.* «Clustering: Multiply and Conquer», Data Communications, JuL 1997.

66. *Briggs, F.* «Synchronization, Coherence, and Event Ordering in Multiprocessors», IEEE Computer, Feb. 1988, pp. 9-21.

67. *Brumе, L, Lefevre, L, Reymann, O.* «Execution Analysis of DSM Applications: A Distributed and Scalable Approach», Proceedings SPDT'96, ACM S1GMETRICS Symposium on Parallel and Distributed Tools, Philadelphia, PA, USA, May 1996, pp. 51–60.

68. *Colder, B., Grunwald, D.* «Fast & Accurate Instruction Fetch and Branch Prediction», ACM SIGARCH Computer Architecture News, Proceedings of the 21st Annual International Symposium on Computer Architecture, Vol. 24, issue 2, Apr. 1994, pp. 2–11.

69. *Can, R. W.* «Virtual Memory Management», UMI Research Press, Annual Arbor, Michigan, 1984.

70. *Chang, P. Y.. Hao, E, Yeh, T. Y., Patt, Y.* «Branch Classification: a New Mechanism for Improving Branch Predictor Performance», Proceedings of the 27'" ACM/IEEE International Symposium on Microarchitecture, Dec. 1994, pp. 22-31.

71. *Chen, P.* «Interconnection Networks Using Shuffles», IEEE Computer, Vol. 14, 1981, pp. 55-64.

72. *Cheong, H., Veldenbaum, A.* «Software-directed Cache Management in Multiprocessors», Cache and Interconnect Architectures in Multiprocessors. 1990, pp. 259-276.

73. *Chong, Y. M.* «Data Flow Chip Optimizes Image Processing», Computer Design. Oct. 1984.PP.97-103.

74. *Clark, D., Levy, H.* «Measurement and Analysis of Instruction Use in the VAX-11/780», Proceedings of the 9th Annual Symposium on Computer Architecture. Apr. 1982, pp. 9-17.

75. *Cline, B.* * Microprogramming Concepts and Techniques», New York: Petrocelli, 1981.

76. *Clos, C.* «A Study of Non-Blocking Switching Networks». Bell Systems Technical Journal, Mar. 1953. pp. 406-424.

77. *Cocke, J., Sweeney D. W.* «High Speed Arithmetic in a Parallel Device», Technical Report IBM, Feb. 1957.

78. *Coffman, E. G.,Jr., ed.* «Computer and Job-Shop Scheduling Theory», John Wiley and Sons, 1976. 141 pp.

79. *Colwell, R. P., Nix, R. P., O'Donnell, J. J., Papworth, D. B., Rodman, P. K.* «A VLIW Architecture for a Trace Scheduling Compiler». Proceedings of the 2nd International Conference on Architectural Support for Programming Languages and Operating Systems, Oct. 1987, pp. 180-192.

80. *Comte, D., Hifdi, N.* «LAU Multiprocessor Microfunctional Description and Technologic Choice», Proceedings of the 1st Europe Conference on Parallel and Distributed Processing, Feb. 1979. pp. 8–15.

81. *Comte, D., Hifdi, N.. Syre, J. C.* «The Data Driven LAU Multiprocessor System: Results and Perspectives», Proceedings World Comp. Congress IF1P '80, Oct. 1980, pp. 175-180.

82. *Cornish, M.* «The TI Dataflow Architecture: The Power of Concurrency for Avionics», Proceedings of the 3rd Conference on Digital Avionics Systems, Nov. 1979, pp. 19-25.

83. *Curnow, H.J., Wichmann, B. A.* «A Synthetic Benchmark», Computer Journal, Vol. 19, № 1, 1976, pp. 43-49.

84. *Dasgupta, S.* «A Hierarchical Taxonomic System for Computer», Computer, Vol. 23, № 3, 1990, pp. 64-74.

85. *Davis, A. L.* «The Architecture and System Method of DMM1: A Recursively Structured Data Driven Machine». Proceedings 5th ISCA, Apr. 1978, pp. 210-215.

86. *Dias, D. M., Kumar, M.* «Packet Switching in nlogn Multistage Networks», Proceedings of the GLOBECOM '84, Atlanta, 1984, pp. 114–120.

87. *Denning, P.* «The Working Set Model for Program Behaviour». Communications of the ACM, May 1968, pp. 323-333.

88. *Denning, P.* «Virtual Memory», Computing Surveys, Vol. 2, Sep. 1970, pp. 153–189.

89. *Dennis, J. B., Misunas, D. P.* «A Preliminary Architecture for a Basic Dataflow Processor», Proceedings of the 2nd Annual Symposium on Computer Architecture, 1975, pp. 126-132.

90. *Dennis, J. B.* «Dataflow Supercomputers», IEEE Computer. № 13, 1980, pp. 48-56.

91. *Doetting, G.,* et al. «S/390 Parallel Enterprise Server Generation 3: A Balanced System and Cache Structure», IBM Journal of Research and Development, Jul./Sep. 1997.

92. *Dongarra, J.* «The UNPACK Benchmark: An Explanation», SuperComputing, Spring 1988, pp. 10-14.

93. *Dubois, M., Scheurich, C., Briggs, F. A.* «Synchronization, Coherence, and Event Ordering in Multiprocessors», Computer, Vol. 21, № 2, Feb. 1988, pp. 9-21.

94. *Duncan, R.* «A Survey of Parallel Computer Architectures», Computer. Vol. 23, № 2, 1990. pp. 5-16.

95. *Evers,* P., *Chang, C, Patt, Y.* «Using Hybrid Branch Predictors to Improve Branch Prediction Accuracy in the Presence of Context Switches», Proceedings of the 23rd International Symposium on Computer Architecture, May 1996, pp. 3–11.

96. *Feng T.* «Some Characteristics of Associative Parallel Processing», Proceedings 1972 Sagamore Computing Conference, 1972, pp. 5-16.

97. *Fleming, P. J., Wallace, J. J.* «How not to Lie with Statistics: the Correct Way to Summarize Benchmark Results», CACM, 29(3), Mar. 1986, pp. 218-221.

98. *Flood, J. E.* «Telecommunications Switching. Traffic and Networks», Prentice-Hall, 1995.

99. *Flynn, M.J.* «Very High-Speed Computing System», Proceedings IEEE, № 54, 1966, pp. 1901-1909.

100. *Flynn, M.J.* «Some Computer Organizations and their Effectiveness», IEEE Transactions on Computers, Vol. 24, Sep. 1972, pp. 948-960.

101. *Flynn, M.J.* «Parallel Processors Were the Future... and May Yet Be», IEEE Computer, Vol. 29, №. 12, Dec. 1996, pp. 151-152.

102. *Fox, G. C.* «What have we learned from using Real Parallel Machines to Solve Real Problems?», The 3rd Conference on Hypercube Concurrent Computers and Applications, Vol. 2, Jan. 1988, pp. 897-955.

103. *Friedman, H. P.* «Statistical Methods in Computer Performance Evaluation. Experimental Computer Performance Evaluation», North-Holland, 1981, pp. 79-105.

104. *Gehringer, E. F., Schwetman, H. G.* «Run-time characteristics of a Simulation Model» Proceedings Simposium on the Simulation Computer System, 1984, pp. 121—129.

105. *Geist, G. A., Sunderam, V. S.* «Network-based Concurrent Computing on the PVM System», Concurrency — Practice & Experience, Vol. 4, № 4, 1992, pp. 293–311.

106. *Giloi, W. K.* «Towards a Taxonomy of Computer Architecture Based on the Machine Data Type View», Proceedings of the 10th Annual International Symposium on Computer Architecture, 1983, pp. 6-17.

107. *Gloy, N.*. et al. «An Analysis of Dynamic Branch Prediction Schemes on System Workloads», ACM SIGARCH Computer Architecture News, Proceedings of the 23rd Annual International Symposium on Computer Architecture, May 1996, pp. 12-21.

108. *Goldstine, H.* «The Computer from Pascal to von Neumann», Princeton University Press, 1972.

109. *Goke, L. R., Lipovski, G.J.* «Banyan Networks for Partitioning Multiprocessor Systems», Proceedings of the 1st Annual Symposium on Computer Architecture, 1973, pp. 21-28.-

110. *Goodman, J. R.* «Using Cache Memory to Reduce Processor-Memory Traffic», Proceedings of the 10th International Symposium on Computer Architecture, 1983.

111. *Goor, A.* «Computer Architecture and Design», Reading, MA: Addison-Wesley, 1989.

112. *Goosen, H., Cheriton, D.* «Predicting the Performance of Shared Memory Caches», Cache and Interconnect Architectures in Multiprocessors, 1990, pp. 153-164.

113. *Gupta, A., Weber, W.* «Analysis of Cache Invalidation Patterns in Shared-Memory Multiprocessors», Cache and Interconnect Architectures in Multiprocessors, 1990, pp. 83-107.

114. *Gurd, J. R.* The Manchester dataflow machine. Future Generations Computer Systems 1, 1985, pp. 201-212.

115. *Gustafson, J.L.* «Reevaluating Amdahl's Law», CACM, 31(5), 1988, pp. 532-533.

116. *Hayes, J. P.* «Computer Architecture and Organization», 2nd International Edition, McGraw-Hill Book Company, Singapore, 1988.

117. *Handler, W.* «On Classification Schemes for Computer Systems in the Post von Neumann Era», Lecture Notes in Computer Science, 1975.

118. *Handler, W.* «The Impact of Classification Schemes on Computer Architecture», Proceedings 1977 ICPP, 1977, pp. 7-15.

119. *Hennessy, J. L.* «VLSI Processor Architecture», IEEE Transactions on Computers, Dec. 1984.

120. *HennessyJ. L., Patterson, D. A.* «Computer Architecture: A Quantitative Approach», 2nd Edition, Morgan Kaufmann Publishers, San Francisco, CA, USA, 1996.

121. *Higbie, L. C.* «Supercomputer architecture», Computer, Vol. 6, № 12, 1973, pp. 48-56.

122. *Hill, M.* «Evaluating Associativity in CPU Caches», IEEE Transactions on Computers, Dec. 1989.

123. *Hiraki, K., Shimada, T., Nishida, K.* A hardware design of the SIGMA-1, a dataflow computer for scientific computations, Proceedings 1984 ICCP, Aug. 1984, pp. 524-531.

124. *Hiraki, K., Sekiguchi, S., Shimada, T.* «Status report of SIGMA-1: A Dataflow Supercomputer, Advanced Topics in Data-Flow Computing» (Gaudiot J-L., and Bic L., eds), Prentice-Hall, 1991, pp. 207-223.

125. *Hockney, R.* «Parallel Computers: Architecture and Performance», Proceedings of International Conference on Parallel Computing '85, 1986, pp. 33-69.

126. *Hockney, R,* «Classification and Evaluation of Parallel Computer Systems», Lecture Notes in Computer Science, № 295, 1987, pp. 13-25.

127. *Huang, A., Knauer, S.* «STARLITE: A Wideband Digital Switch», Proceedings of the GLOBECOM '84, Atlanta, 1984, pp. 121-125.

128. *Huck, T.* «Comparative Analysis of Computer Architectures», Stanford University Technical Report, № 83-243, May 1983.

129. *Huber, M. N., Frantzen, V., Maegerl, G.* «Proposed Evolutionary Paths for B-ISDN Signalling», Proceedings of the XIV International Switching Symposium, Yokohama, 1992, paper C3.3.

130. *Hwang, K., Briggs, F.A.* «Computer Architecture and Parallel Processing», McGraw-Hill, 1984.

131. *Hwang, K., Xu, 2.* «Scalable Parallel Computing», McGraw-Hill, 1998.

132. *Hwang, K.* «Advanced Computer Architecture: Parallelism, Scalability, Programinability», New York: McGraw-Hill Inc., 1993.

133. IEC 60027-2 «Letter Symbols to be Used in Electrical Technology - Part2», Telecommunication and Electronics, Nov. 2000.

134. Inmos Ltd. «Transputer Reference Manual», Bristol, England, 1986.

135. *Iwashita, M.*, et.al. «Modular Dataflow Image Processor», Proceedings COMPCON Fall '83, 1983, pp. 464-467.

136. *Jacob, B., Mudge, T.* «Notes on Calculating Performance», University of Michigan Technical Report CSE-TR-231-95, Mar. 1995, pp. 1-10.

137. *Jain, R.* «The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling», New York: Wiley-Interscience, Apr. 1991, 720 pp.

138. *James, D. V.* «SCI (Scalable Coherent Interface) Cache Coherence», Cache and Interconnect Architectures in Multiprocessors, 1990, pp. 189-208.

139. *Johnson, E. E.* «Completing an MIMD Multiprocessor Taxonomy», Computer Architecture News, Vol. 16, № 2,1988, pp. 44-48.

140. *Jones, S. A.* «Futurebus Interface from Off-the-ShelfParts», IEEE Micro, Feb. 1991.

141. *Katz, R. H., Eggers, S.J., Wood, D. A., Perkins, C. L., Sheldon, R. G.* «Implementing a Cache Consistency Protocol», Proceedings of the 12th International Symposium on Computer Architecture, 1985.

142. *Kieburtz, R. B.* «The G Machine: A Fast, Graph-Reduction Evaluator», LNCS 201, Springer Verlag. 1985, pp. 400-413.

143. *Kishi, M., Yasuhara, #., Kawamura, Y.* «DDDP: A Distributed Data Driven Processor», Proceedings 10thISCA, Jun. 1983, pp. 236-242.

144. *Knudsen, M. T.* «PMSL: An Interactive Language for The System Level Description and Analysis of Computer Structures», Carnegie-Mellon University Press. 1975.

145. *Krishnamurthy, E. V.* «Parallel Processing Principles and Practice», Addison-Wesley. 1989, pp. 208-246.

146. *Kumar, M.* «Measuring Parallelism in Computation-Intensive Scientific/Engineering Applications», IEEE Transactionson Computers, Vol. 37, № 9, Sept. 1988, pp. 1088-1098.

147. *Kung, S. Y.* «On Supercomputing with Systolic/Wavefront Array Processors», Proceedings IEEE, Vol. 72, № 47, 1984, pp. 867-884.

148. *Lawrie, D. H.* «Access and Alignmentof Data in anArray Processor», IEEETransactions on Computers, C-24, № 12, 1975, pp. 1145-1155.

149. *Lea, C. T.* «Multi-$\log_2$N Self-Routing Networks and their Applications in High Speed Electronic and Photonic Switching Systems», Proceedings of the INFO COM'89, Ottawa, 1989, pp. 877–886.

150. *Lee, J., Smith, A.* «Branch Prediction Strategies and Branch Target Buffer Design», IEEE Computer, Vol. 17 (1), Jan. 1984, pp. 6-21.

151. *Lehman, M.* «High-speed Digital Multiplication», IRE Transaction on Electronic Computers, Vol. EC-6–6, № 3, 1957.

152. *Lenfant, J.* «Parallel Permutations of Data: A Benes Network Control Algorithms for Frequently Used Permutations», IEEE Transactions on Computers, C-27, № 7, 1978, pp. 637-647.

153. *Li, K., Hudak, P.* «Memory Coherence in Shared Virtual Memory Systems»', ACM Transactions on Computer Systems, Nov. 1989, pp. 321-357.

154. *Lilja, D.* «Reducing the Branch Penalty in Pipelined Processors», Computer, Jul. 1988.

155. *Lilja, D.J.* «Cache Coherence in Large-Scale Shared-Memory Multiprocessors: Issues and Comparison», ACM Computing Surveys, 25 (3), Sep. 1993.

156. *Lipovski, G. L.* «Banyan Networks for Partitioning Multiprocessor Systems», Proceedings of the 1" International Symposium on Computer Architecture, 1973, pp. 21-28.

157. *Lovett, T., Clapp, R.* «Implementation and Performance of a CC-NUMA System», Proceedings of the 23rd Annual International Symposium on Computer Architecture, May 1996.

158. *Lunde, A.* «Empirical Evaluation of Some Features of Instruction Set Processor Architectures», Communications of the ACM, Mar. 1977.

159. *Mak, P.,* et al. «Shared-Cache Clusters in a System with a Fully Shared Memory», IBM Journal of Research and Development, Jul./Sep. 1997.

160. *Mamrak, S. A., Amer, P. D.* «Statistical procurement methodologies. Experimental computer performance evaluation», North-Holland, 1981, pp. 118–132.

161. MANO95 *Mano, M.M.* «Computer System Architecture», 3rd Edition. LA Prentice-Hall, 1995.

162. *Mono, T.* «Cache Coherence for Scalable Shared Memory Multiprocessors», Technical Report, Computer System Laboratory, Stanford University, May 1992.

163. *Massiglia, P.,* ed. «The RAIDbook: A Sourcebook for Disk Array Technology», St. Peter, MN: The RAID Advisory Board, 1994.

164. *Mayberry, W., Efland, G.* «Cache Boosts Multiprocessor Performance», Computer Design, Nov. 1984.

165. *McFarling* «Combining Branch Predictors», WRL Technical Note TN-36, Digital Equipment Corporation, Jun., 1993.

166. Message Passing Interface Forum «MPI: A Message-Passing Interface Standard — Version 1.1», http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html/, Jun 1995.

167. Message Passing Interface Forum «MPI-2: Extensions to the Message-Passing Interface», http://www.mcs.anl.gov/mpi/, Jun. 1997.

168. *Moor, G.* «Cramming More Components onto Integrated Circuits», Electronics, Vol. 38, № 7, Apr, 19, 1965, pp. 114–117.

169. *Mou, Z., Jutand, F.* «"Overturned Stairs" Adder Trees and Multiplier Design», IEEE Transactions on Computers, C-41, Apr. 1992, pp. 940-948.

170. *Nair, R.* «Effect of Increasing Chip Density on the Evolution of Computer Architectures», IBM Journal of Research and Development, Vol. 46, № 2/3, May 2002, pp. 223-234.

171. *Narasimhan, V. L., Downs, T.* «Operating System Features of a Dynamic Dataflow Array Processing System (PATTSY)», Proceedings of the 3rd Annual Parallel Processing Symposium, Mar. 1989, pp. 722-740.

172. *Nassimi, D.* «A Self-Routing Benes Network and Parallel Permutation Algorithms», IEEE Transactions on Computers, C-30, №5, 1981, pp. 332-340.

173. *Ni, L M., McKinley, P. K.* «A Survey of Wormhole Routing Techniques in Direct Networks», IEEE Computer, Vol. 26, № 2, Feb. 1993, pp. 62-76.

174. An Overview Of Computational Science, http://csep1.phy.ornl.gov/ov/ov.html.

175. *Papamarcos, M. S., Patel, J. H.* «A Low-Overhead Coherence Solution for Multiprocessors with Private Cache Memories», Proceedings of the 11th International Symposium on Computer Architecture, 1984.

176. *Patel, J. H.* «Performance of Processor-Memory Interconnection for Multiprocessors», IEEE Transactions on Computers, Vol. 30, № 10, Oct. 1981, pp. 881-780.

177. *Patterson, D., Ditzel, D.* «The case for the reduced instruction set computer», Computer Architecture News, Vol. 8, № 6, Oct 1980, pp. 25-33.

178. *Patterson, D.* «Reduced Instruction Set Computers», Communications of the ACM, Jan. 1985, pp. 8-21.

179. *Patterson, D. A., Garth,* ft., *Katz, R.* «A Case for Redundant Arrays of Inexpensive Disks (RAID)», University of California, Berkeley, Report № UCB SCD/87/391, Dec. 1987.

180. *Pease, l. M. C.* «The Indirect Binary n-Cube Microprocessor Array», IEEE Transactions on Computers, C-26, № 5, 1977, pp. 458-473.

181. *Pezaris, S. D.* «A 40-ns 17b by 17b Array Multiplier», IEEE Transactions on Computers, C- 20, Apr. 1971, pp. 442-447.

182. *Pfister, G.* «In Search of Clusters», Upper Saddle River, New Jersey: Prentice-Hall, 1998.

183. *Przybylski, S.* «The Performance Impact of Block Size and Fetch Strategies», Proceedings of the 17th Annual International Symposium on Computer Architecture, May 1990.

184. *Robertson, J. E.* «A New Class of Digital Division Methods», IEEE Transactions on Computers, Electronic Computers, EC-7, Sep. 1958, pp. 218-222.

185. International Roadmap for Semiconductors, http://public.itrs.net.

186. *Rudolph, L., Segall, Z.* «Dynamic Decentralized Cache Schemes for MIMD Parallel Processors», Proceedings of the 11th International Symposium on Computer Architecture, 1984.

187. *Shore, J. E.* «Second Thoughts on Parallel Processing», Comput. Elect. Eng., № 1, 1992, pp. 95-109.

188. *Sieget, H.J.* «The Multistage Cube: A Versatile Interconnection Network», IEEE Computer, Vol. 14, № 12, 1981, pp. 65-76.

189. *Siemorek, D. P.* «Introduction to PMS», Computer, Vol 7, № 12, 1974, pp. 42-44.

190. *Siemorek, D. P., Bell, C.G., Newell, A.* «Computer Structures: Principles and Examples», New York: McGraw-Hill, 1982, 923 pp.

191. *Sima, D., Fountain, T.,* Kacsuk. P. «Advanced Computer Architectures — a Design Space Approach», Addison-Wesley, 1997, 766 pp.

192. *Skillicorn, D. A.* «Taxonomy for Computer Architectures», Computer, Vol. 21, № 11, 1988, pp. 46-57.

193. *Smith, J.* «A Study of Branch Prediction Strategies», ISCA, Proceedings of the 8th Annual International Symposium on Computer Architecture (selected papers), May 1981.

194. *Smith, A.* «Cache Memories», ACM Computing Surveys, Sep. 1982.

195. *Smith, A.* «Line (Block) Size Choice for CPU Cache Memories», IEEE Transactions on Communications, Sep. 1987.

196. *Smith, J. E.* «Characterizing Computer Performance with a Single Number», CACM, 31(10), Oct. 1988, pp. 1202-1206.

197. *Smith, J. E.* «A Study of Branch Prediction Strategies», ISCA, ACM, 25 Years of the International Symposium on Computer Architecture (selected papers), Aug. 1998.

198. *Smiling, D. F.* «The Design and Analysis of a Stateless Data-Flow Architectures», Technical Report UMCS-93-7-2, University of Manchester, Department of Computer Science, 1993.

199. *Snyder, L.* «A Taxonomy of Synchronous Parallel Machines», University Park, Pennsylvania, 1988, pp. 281-289.

200. *Stallings, W.* «Computer Organization and Architecture», 5th Edition, Prentice-Hall, 1999.

201. *Stenstrum, P.* «A Surveyof Cache Coherence Schemes for Multiprocessors», IEEE Computer, Jun. 1990, pp. 12-24.

202. *Stoy, J.E.* «Denotational Semantics», MIT Press, 1981.

203. *Stone, H. S.* «Parallel Processing with the Perfect Shuffle», IEEE Transactions on Computers, C-20, № 2, 1971, pp. 153-161.

204. *Sun, X. H., Ni, L. M.* «Scalable Problems and Memory-Bounded Speedup*. Journal of Parallel and Distributed Computing, № 19, 1993, pp. 27-37.

205. *Tabak, D.* «RISC Systems»,John Wiley & Sons Inc., 1990.

206. *Takahashi, N., Amamija, M.* «A Data Flow Processor Array System: Design and Analysis», Proceedings 10* ISCA, Jun. 1983, pp. 243-250.

207. *Talcott, A. R., Yamamoto, W.. Serrano, M.J., Wood, R. C, Nemirovsky, V.* «The Impac of Unresolved Branches on Branch Prediction Scheme Performance», ACM SIGARCH Computer Architecture News, Proceedings of the 2Г Annual International Symposium on Computer Architecture, Vol. 24, issue 2, Apr. 1994, pp. 12-21.

208. *Tamic, Y., Sequin, C.* «Strategies for Managing the Register File in RISC», IEEE Transactions on Computers, Nov. 1983.

209. *Tanenbaum, A.* «Implications of Structured Programming for Machine Architecture», Communications of the ACM, Mar. 1978.

210. *Tanenbaum. A.* «Structured Computer Organization», 3rd edition, Prentice-Hall International, 1990[1].

211. *Thacker, C. P., Stewart, L. C.Satterthwaite, E. H.*« Firefly: A Multiprocessor Workstation», IEEE Transactions on Computers. Vol. 37, № 8, Aug. 1988, pp. 909-920.

212. *Thakkar, $., Dubios, M., Laundrie, A. T.,Sohi, G. S.* «Scalable Shared-Memory Multiprocessor Architectures», IEEE Computer, 23 <6), Jun. 1990, pp. 71-83.

213. *Thurber, K.J.* «Large Scale Computer Architecture», Hayden Book Company, Rochelle Park, New Jersey, 1976.

214. *Tocher, K. D.* «Techniques of Multiplication and Division for Automatic Binary Computers», Quart. J. Mech. Appl. Math., 11, Jul./Sep. 1958, pp. 364-384.

215. *Tomasevic, M., Milutinovic, V.* «The Cache Coherence Problem in Shared-Memory Multiprocessors», IEEE Computer Society Press, Los Alamitos, CA. 1993.

216. *Treleaven, P. C, Brownbridge, D. R., Hopkins, R. P.* «Data-Driven and Demand-Driven Computer Architecture», ACM Computing Surveys, 14 (1), Mar. 1982.

[1] Русский перевод: «Архитектура компьютера». 4-е изд, 2003 год. Издательский дом «Питер». - *Примеч. ред.*

217. *Turner, J. S.* «Design of a Broadcast Packet Switching Network», IEEE Transactions on Communications, Vol. 36, № 6, Jun. 1987, pp. 734-743.

218. *Vedder, R., Campbell, M., Tucker, G.* «The Hudges Data Flow Multiprocessor», Proceedings of the 5th International Conference on Distributed Computing Systems, May 1985, pp. 324-332.

219. *Von Neumann, J.* «First Draft of a Report on the EDVAC», Moore School, University of Pennsylvania, 1945.

220. *Wang, J., Dubois, M.* «Memory-Access Penalties in Write-Invalidate Cache Coherence Protocols», Cache and Interconnect Architectures in Multiprocessors, 1990, pp. 109-130.

221. *Weicker, R. P.* «Dhrystone: A Synthetic Systems Programming Benchmark», CACM, 27(10), Oct. 1984, pp. 1013-1030.

222. *Weizer, N,* et al. «The Arthur D. Little Forecast on Information Technology and Productivity», New York: Wiley, 1991.

223. *Whitney, S.,* et al. «The SGI Origin Software Environment and Application Performance», Proceedings COMPCON Spring '97, Feb. 1997.

224. *Wilkes, M.* «The Best Way to Design an Automatic Calculating Machine», Proceedings Manchester University Computer Inaugural Conference, Jul. 1951.

225. *Wilkinson, B.* «Computer Architecture: Design and Performance», New York: Prentice-Hall, 1996.

226. *Williams, F., Steven, G* «Address and Data Register Separation on the M68000 Family», Computer Architecture News, Jun. 1990.

227. *Wu, C. L., Feng, T. Y.* «On a Class of Multistage Interconnection Networks», IEEE Transactions on Computers, Vol. 29, № 8, Aug. 1980, pp. 694-702.

228. *Xie, J., Guo, B.* «An Evaluation and Comparative Analysis of Branch Prediction Schemes on Alpha Processors», Duke University, Department of Computer Science, Dec. 5, 2000.

229. *Yeh, T., Patt, Y. N.* «Two-Level Adaptive Training Branch Prediction», Proceedings of the 24th Annual International Symposium on Microarchitecture, Albuquerque, 1991, pp. 51-61.

230. *Yeh, T., Patt, Y. N.* «Alternative Implementation of Two-Level Adaptive Branch Prediction», Proceedings of the International Symposium on Computer Architecture, 1992, pp. 124-134.

231. *Yeh, T, Patt, Y. N.* «Comparison of Dynamic Branch Predictors that use Two Levels of Branch History», Proceedings of the International Symposium on Computer Architecture, 1993, pp. 257-265.

232. *Young, C., Gloy, N.. Smith, N.* «A Comparative Analysis of Schemes for Correlated Branch Prediction», Proceedings of the International Symposium on Computer Architecture, 1995, pp. 287-295.

233. *Zhou, M., Su, Z.* «A Comparative Analysis of Branch Prediction Schemes», Technical Report, http://www.cs.berkeley.edu/~zhendong/cs252/project.html, University of California, Berkeley, 1995.

234. *Zomaya, Y.* «Parallel and Distributed Computing HandBook», McGraw, 1997.

Алфавитный указатель

В

Г

Д

Е

Ж

З

И

Р

С

Т

У

Ф

Цилькер Борис Яковлевич,
Орлов Сергей Александрович
Организация ЭВМ и систем: Учебник для вузов

Главный редактор	*Е. Строганова*
Заведующий редакцией	*И. Корнеев*
Руководитель проекта	*Ю. Суркис*
Литературный редактор	*Е. Васильев*
Художник	*Н. Биржаков*
Корректор	*В. Листом*
Верстка	*Л. Харитонов*

Лицензия ИД № 05784 от 07.09.01.

Подписано печати 17.12.03. Формат 70×100/16. Усл. а. а. 54.18.
Тираж 4500. Заказ 470

ООО «Питер Принт». 196105, Санкт-Петербург, ул. Благодатная, д. 67.

Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.

Отпечатано с готовых диапозитивов в ФГУП ордена Трудового Красного Знамени
«Техническая книга» Министерств* Российской Федерации
по делам печати, телерадиовещания и средств массовых коммуникаций
190005, Санкт-Петербург, Измайловский пр., 29