Московский физико-технический институт
Факультет прикладной математики и экономики
Кафедра "Системное программирование"

Шнитман В.З.

АРХИТЕКТУРА СОВРЕМЕННЫХ КОМПЬЮТЕРОВ

Учебное пособие

2008

ОГЛАВЛЕНИЕ

1. 1. Общие требования, предъявляемые к современным компьютерам

1.1. 1.1. Отношение стоимость/производительность

Появление любого нового направления в вычислительной технике определяется требованиями компьютерного рынка. Поэтому у разработчиков компьютеров нет одной единственной цели. Большая универсальная вычислительная машина (мейнфрейм) или суперкомпьютер стоят дорого. Для достижения поставленных целей при проектировании высокопроизводительных конструкций приходится игнорировать стоимостные характеристики. Суперкомпьютеры фирмы Cray Research и высокопроизводительные мейнфреймы компании IBM относятся именно к этой категории компьютеров. Другим крайним примером может служить низкостоимостная конструкция, где производительность принесена в жертву для достижения низкой стоимости. К этому направлению относятся персональные компьютеры различных клонов IBM PC. Между этими двумя крайними направлениями находятся конструкции, основанные на отношении стоимость/ производительность, в которых разработчики находят баланс между стоимостными параметрами и производительностью. Типичными примерами такого рода компьютеров являются миникомпьютеры и рабочие станции.

Для сравнения различных компьютеров между собой обычно используются стандартные методики измерения производительности. Эти методики позволяют разработчикам и пользователям использовать полученные в результате испытаний количественные показатели для оценки тех или иных технических решений, и в конце концов именно производительность и стоимость дают пользователю рациональную основу для решения вопроса, какой компьютер выбрать.

1.2. 1.2. Надежность и отказоустойчивость

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

Отказоустойчивость - это такое свойство вычислительной системы, которое обеспечивает ей, как логической машине, возможность продолжения действий, заданных программой, после возникновения неисправностей. Введение отказоустойчивости требует избыточного аппаратного и программного обеспечения. Направления, связанные с предотвращением неисправностей и с отказоустойчивостью, - основные в проблеме надежности. Концепции параллельности и отказоустойчивости вычислительных систем естественным образом связаны между собой, поскольку в обоих случаях требуются дополнительные функциональные компоненты. Поэтому, собственно, на параллельных вычислительных системах достигается как наиболее высокая производительность, так и, во многих случаях, очень высокая надежность. Имеющиеся ресурсы избыточности в параллельных системах могут гибко использоваться как для повышения производительности, так и для повышения надежности. Структура многопроцессорных и многомашинных систем приспособлена к автоматической реконфигурации и обеспечивает возможность продолжения работы системы после возникновения неисправностей.

Следует помнить, что понятие надежности включает не только аппаратные средства, но и программное обеспечение. Главной целью повышения надежности систем является целостность хранимых в них данных.

1.3. 1.3. Масштабируемость

Масштабируемость представляет собой возможность наращивания числа и мощности процессоров, объемов оперативной и внешней памяти и других ресурсов вычислительной системы. Масштабируемость должна обеспечиваться архитектурой и конструкцией компьютера, а также соответствующими средствами программного обеспечения.

Добавление каждого нового процессора в действительно масштабируемой системе должно давать прогнозируемое увеличение производительности и пропускной способности при приемлемых затратах. Одной из основных задач при построении масштабируемых систем является минимизация стоимости расширения компьютера и упрощение планирования. В идеале добавление процессоров к системе должно приводить к линейному росту ее производительности. Однако это не всегда так. Потери производительности могут возникать, например, при недостаточной пропускной способности шин из-за возрастания трафика между процессорами и основной памятью, а также между памятью и устройствами ввода/вывода. В действительности реальное увеличение производительности трудно оценить заранее, поскольку оно в значительной степени зависит от динамики поведения прикладных задач.

Возможность масштабирования системы определяется не только архитектурой аппаратных средств, но зависит от заложенных свойств программного обеспечения. Масштабируемость программного обеспечения затрагивает все его уровни от простых механизмов передачи сообщений до работы с такими сложными объектами как мониторы транзакций и вся среда прикладной системы. В частности, программное обеспечение должно минимизировать трафик межпроцессорного обмена, который может препятствовать линейному росту производительности системы. Аппаратные средства (процессоры, шины и устройства ввода/вывода) являются только частью масштабируемой архитектуры, на которой программное обеспечение может обеспечить предсказуемый рост производительности. Важно понимать, что простой переход, например, на более мощный процессор может привести к перегрузке других компонентов системы. Это означает, что действительно масштабируемая система должна быть сбалансирована по всем параметрам.

1.4. 1.4. Совместимость и мобильность программного обеспечения

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

В настоящее время одним из наиболее важных факторов, определяющих современные тенденции в развитии информационных технологий, является ориентация компаний-поставщиков компьютерного оборудования на рынок прикладных программных средств. Это объясняется прежде всего тем, что для конечного пользователя в конце концов важно программное обеспечение, позволяющее решить его задачи, а не выбор той или иной аппаратной платформы. Переход от однородных сетей программно совместимых компьютеров к построению неоднородных сетей, включающих компьютеры разных фирм-производителей, в корне изменил и точку зрения на саму сеть: из сравнительно простого средства обмена информацией она превратилась в средство интеграции отдельных ресурсов - мощную распределенную вычислительную систему, каждый элемент которой (сервер или рабочая станция) лучше всего соответствует требованиям конкретной прикладной задачи.

Этот переход выдвинул ряд новых требований. Прежде всего такая вычислительная среда должна позволять гибко менять количество и состав аппаратных средств и программного обеспечения в соответствии с меняющимися требованиями решаемых задач. Во-вторых, она должна обеспечивать возможность запуска одних и тех же программных систем на различных аппаратных платформах, т.е. обеспечивать мобильность программного обеспечения. В третьих, эта среда должна гарантировать возможность применения одних и тех же человеко-машинных интерфейсов на всех компьютерах, входящих в неоднородную сеть. В условиях жесткой конкуренции производителей аппаратных платформ и программного обеспечения сформировалась концепция открытых систем, представляющая собой совокупность стандартов на различные компоненты вычислительной среды, предназначенных для обеспечения мобильности программных средств в рамках неоднородной, распределенной вычислительной системы.

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

2. 2. Классификация компьютеров по областям применения

1.5. 2.1. Персональные компьютеры и рабочие станции

Персональные компьютеры (ПК) появились в результате эволюции миникомпьютеров при переходе элементной базы машин с малой и средней степенью интеграции на большие и сверхбольшие интегральные схемы. ПК, благодаря своей низкой стоимости, очень быстро завоевали хорошие позиции на компьютерном рынке и создали предпосылки для разработки новых программных средств, ориентированных на конечного пользователя. Это прежде всего - "дружественные пользовательские интерфейсы", а также проблемно-ориентированные среды и инструментальные средства для автоматизации разработки прикладных программ.

Миникомпьютеры стали прародителями и другого направления развития современных систем - 32-разрядных машин. Создание RISC-процессоров и микросхем памяти емкостью более 1 Мбит привело к окончательному оформлению настольных систем высокой производительности, которые сегодня известны как рабочие станции. Первоначальная ориентация рабочих станций на профессиональных пользователей (в отличие от ПК, которые в начале ориентировались на самого широкого потребителя непрофессионала) привела к тому, что рабочие станции - это хорошо сбалансированные системы, в которых высокое быстродействие сочетается с большим объемом оперативной и внешней памяти, высокопроизводительными внутренними магистралями, высококачественной и быстродействующей графической подсистемой и разнообразными устройствами ввода/вывода. Это свойство выгодно отличает рабочие станции среднего и высокого класса от ПК и сегодня.

Тем не менее, быстрый рост производительности ПК на базе новейших микропроцессоров Intel в сочетании с резким снижением цен на эти изделия и развитием технологии локальных шин (VESA и PCI), позволяющей устранить многие "узкие места" в архитектуре ПК, делают современные персональные компьютеры весьма привлекательной альтернативой рабочим станциям. В свою очередь производители рабочих станций создали изделия так называемого "начального уровня", которые по стоимостным характеристикам близки к высокопроизводительным ПК, но все еще сохраняют лидерство по производительности и возможностям наращивания. Насколько успешно удастся ПК на базе процессоров Pentium бороться против рабочих станций UNIX, покажет будущее, но уже в настоящее время появилось понятие "персональной рабочей станции", которое объединяет оба направления.

Современный рынок "персональных рабочих станций" не просто определить. По сути, он представляет собой совокупность архитектурных платформ персональных компьютеров и рабочих станций, которые появились в настоящее время, поскольку поставщики компьютерного оборудования уделяют все большее внимание рынку продуктов для коммерции и бизнеса. Этот рынок традиционно считался вотчиной миникомпьютеров и мейнфреймов, которые поддерживали работу настольных терминалов с ограниченным интеллектом. В прошлом персональные компьютеры не были достаточно мощными и не располагали достаточными функциональными возможностями, чтобы служить адекватной заменой подключенных к главной машине терминалов. С другой стороны, рабочие станции на платформе UNIX были очень сильны в научном, техническом и инженерном секторах и были почти также неудобны, как и ПК для того чтобы выполнять серьезные офисные приложения. С тех пор ситуация изменилась коренным образом. Персональные компьютеры в настоящее время имеют достаточную производительность, а рабочие станции на базе UNIX имеют программное обеспечение, способное выполнять большинство функций, которые стали ассоциироваться с понятием "персональной рабочей станции". Вероятно,

оба этих направления могут серьезно рассматриваться в качестве сетевого ресурса для систем масштаба предприятия. В результате этих изменений практически ушли со сцены старомодные миникомпьютеры с их патентованной архитектурой и использованием присоединяемых к главной машине терминалов. По мере продолжения процесса разукрупнения (downsizing) и увеличения производительности платформы Intel наиболее мощные ПК (но все же чаще открытые системы на базе UNIX) стали использоваться в качестве серверов, постепенно заменяя миникомпьютеры.
Среди других факторов, способствавших этому процессу, следует выделить:

- • Применение ПК стало более разнообразным. Помимо обычных для этого класса систем текстовых процессоров, даже средний пользователь ПК может теперь работать сразу с несколькими прикладными пакетами, включая электронные таблицы, базы данных и высококачественную графику.
- • Адаптация графических пользовательских интерфейсов существенно увеличила требования пользователей ПК к соотношению производительность/стоимость. И хотя оболочка MS Windows могла работать на моделях ПК 386SX с 2 Мбайтами оперативной памяти, реальные пользователи хотели бы использовать все преимущества подобных систем, включая возможность комбинирования и эффективного использования различных пакетов.
- • Широкое распространение систем мультимедиа прямо зависит от возможности использования высокопроизводительных ПК и рабочих станций с адекватными аудио - и графическими средствами, и объемами оперативной и внешней памяти.
- • Слишком высокая стоимость мейнфреймов и даже систем среднего класса помогла сместить многие разработки в область распределенных систем и систем клиент-сервер, которые многим представляются вполне оправданной по экономическим соображениям альтернативой. Эти системы прямо базируются на высоконадежных и мощных рабочих станциях и серверах.

В начале представлялось, что необходимость сосредоточения высокой мощности на каждом рабочем месте приведет к переходу многих потребителей ПК на UNIX-станции. Это определялось частично тем, что RISC-процессоры, использовавшиеся в рабочих станциях на базе UNIX, были намного более производительными по сравнению с CISC-процессорами, применявшимися в ПК, а частично мощностью системы UNIX по сравнению с MS-DOS и даже OS/2.
Производители рабочих станций быстро отреагировали на потребность в низкостоимостных моделях для рынка коммерческих приложений. Потребность в высокой мощности на рабочем столе, объединенная с желанием поставщиков UNIX-систем продавать как можно больше своих изделий, привела такие компании как Sun Microsystems и Hewlett Packard на рынок рабочих станций для коммерческих приложений. И хотя значительная часть систем этих фирм все еще ориентирована на технические приложения, наблюдается беспрецедентный рост продаж продукции этих компаний для работы с коммерческими приложениями, требующими все большей и большей мощности для реализации сложных, сетевых прикладных систем, включая системы мультимедиа.
Острая конкуренция со стороны производителей UNIX-систем и потребности в повышении производительности огромной уже инсталлированной базы ПК, заставили компанию Intel форсировать разработку высокопроизводительных процессоров семейства 486 и Pentium. Процессоры 486 и Pentium, при разработке которого были использованы многие подходы, применявшиеся ранее только в RISC-процессорах, а также использование других технологических усовершенствований, таких как

архитектура локальной шины, позволили снабдить ПК достаточной мощностью, чтобы составить конкуренцию рабочим станциям во многих направлениях рынка коммерческих приложений. Правда, для многих других приложений, в частности, в области сложного графического моделирования, ПК все еще сильно отстают.

1.6. **2.2. X-терминалы**

X-терминалы представляют собой комбинацию бездисковых рабочих станций и стандартных ASCII-терминалов. Бездисковые рабочие станции часто применялись в качестве дорогих дисплеев и в этом случае не полностью использовали локальную вычислительную мощь. Одновременно многие пользователи ASCII-терминалов хотели улучшить их характеристики, чтобы получить возможность работы в многооконной системе и графические возможности. Совсем недавно, как только стали доступными очень мощные графические рабочие станции, появилась тенденция применения "подчиненных" X-терминалов, которые используют рабочую станцию в качестве локального сервера.

На компьютерном рынке X-терминалы занимают промежуточное положение между персональными компьютерами и рабочими станциями. Поставщики X-терминалов заявляют, что их изделия более эффективны в стоимостном выражении, чем рабочие станции высокого ценового класса, и предлагают увеличенный уровень производительности по сравнению с персональными компьютерами. Быстрое снижение цен, прогнозируемое иногда в секторе X-терминалов, в настоящее время идет очевидно благодаря обострившейся конкуренции в этом секторе рынка. Многие компании начали активно конкурировать за распределение рынка, а быстрый рост объемных поставок создал предпосылки для создания такого рынка. В настоящее время уже достигнута цена в $1000 для X-терминалов начального уровня, что делает эту технологию доступной для широкой пользовательской базы.

Как правило, стоимость X-терминалов составляет около половины стоимости сравнимой по конфигурации бездисковой машины и примерно четверть стоимости полностью оснащенной рабочей станции.

Что такое X-терминал?

Типовой X-терминал включает следующие элементы:

- • Экран высокого разрешения - обычно размером от 14 до 21 дюйма по диагонали;
- • Микропроцессор на базе Motorola 68xxx или RISC-процессор типа Intel i960, MIPS R3000 или AMD29000;
- • Отдельный графический сопроцессор в дополнение к основному процессору, поддерживающий двухпроцессорную архитектуру, которая обеспечивает более быстрое рисование на экране и прокручивание экрана;
- • Базовые системные программы, на которых работает система X-Windows, и выполняются сетевые протоколы;
- • Программное обеспечение сервера X11;
- • Переменный объем локальной памяти (от 2 до 8 Мбайт) для дисплея, сетевого интерфейса, поддерживающего TCP/IP и другие сетевые протоколы.
- • Порты для подключения клавиатуры и мыши.

X-терминалы отличаются от ПК и рабочих станций не только тем, что не выполняют функций обычной локальной обработки. Работа X-терминалов зависит от главной (хост) системы, к которой они подключены посредством сети. Для того, чтобы X-терминал мог работать, пользователи должны инсталлировать программное обеспечение многооконного сервера X11 на главном процессоре, выполняющем прикладную задачу (наиболее известная версия X11 Release 5). X-терминалы отличаются также от стандартных алфавитно-цифровых ASCII и традиционных графических дисплейных терминалов тем, что они могут быть подключены к любой главной системе, которая поддерживает стандарт X-Windows. Более того, локальная вычислительная мощь X-терминала обычно используется для обработки отображения, а не обработки приложений (называемых клиентами), которые выполняются удаленно на главном компьютере (сервере). Вывод такого удаленного приложения просто отображается на экране X-терминала.

Минимальный объем требуемой для работы памяти X-терминала составляет 1 Мбайт, но чаще 2 Мбайта. В зависимости от функциональных возможностей изделия оперативная память может расширяться до 32 Мбайт и более.

Оснащенный стандартной системой X-Windows, X-терминал может отображать на одном и том же экране множество приложений одновременно. Каждое приложение может выполняться в своем окне и пользователь может изменять размеры окон, их месторасположение и манипулировать ими в любом месте экрана.

X-Windows - результат совместной работы Масачусетского технологического института (MIT) и корпорации DEC. Система X-Windows (известная также под именем X) в настоящее время является открытым де-факто стандартом для доступа к множеству одновременно выполняющихся приложений с возможностями многооконного режима и графикой высокого разрешения на интеллектуальных терминалах, персональных компьютерах, рабочих станциях и X-терминалах. Она стала стандартом для обеспечения интероперабельности (переносимости) продуктов многих поставщиков и для организации доступа к множеству приложений. В настоящее время X-Windows является стандартом для разработки пользовательского интерфейса. Более 90% поставщиков UNIX-рабочих станций и многие поставщики персональных компьютеров адаптировали систему X-Windows и применяют в качестве стандарта.

1.7. **2.3. Серверы**

Прикладные многопользовательские коммерческие и бизнес-системы, включающие системы управления базами данных и обработки транзакций, крупные издательские системы, сетевые приложения и системы обслуживания коммуникаций, разработку программного обеспечения и обработку изображений все более настойчиво требуют перехода к модели вычислений "клиент-сервер" и распределенной обработке. В распределенной модели "клиент-сервер" часть работы выполняет сервер, а часть пользовательский компьютер (в общем случае клиентская и пользовательская части могут работать и на одном компьютере). Существует несколько типов серверов, ориентированных на разные применения: файл-сервер, сервер базы данных, принт-сервер, вычислительный сервер, сервер приложений. Таким образом, тип сервера определяется видом ресурса, которым он владеет (файловая система, база данных, принтеры, процессоры или прикладные пакеты программ).

С другой стороны существует классификация серверов, определяющаяся масштабом сети, в которой они используются: сервер рабочей группы, сервер отдела или сервер масштаба предприятия (корпоративный сервер). Эта классификация весьма условна. Например, размер группы может меняться в диапазоне от нескольких человек до нескольких сотен человек, а сервер отдела обслуживать от 20 до 150 пользователей. Очевидно, в зависимости от числа пользователей и характера решаемых ими задач требования к составу оборудования и программного обеспечения сервера, к его надежности и производительности сильно варьируются.

Файловые серверы небольших рабочих групп (не более 20-30 человек) проще всего реализуются на платформе персональных компьютеров и программном обеспечении Novell NetWare. Файл-сервер, в данном случае, выполняет роль центрального хранилища данных. Серверы прикладных систем и высокопроизводительные машины

для среды "клиент-сервер" значительно отличаются требованиями к аппаратным и программным средствам.
Типичными для небольших файл-серверов являются: процессор 100 МГц Pentium или более быстродействующий, 32-Мбайт ОЗУ, 2 Гбайт дискового пространства и один адаптер Ethernet 10BaseT, имеющий быстродействие 10 Мбит/с. В состав таких серверов часто включаются флоппи-дисковод и дисковод компакт-дисков. Графика для большинства серверов несущественна, поэтому достаточно иметь обычный монохромный монитор с разрешением VGA.
Скорость процессора для серверов с интенсивным вводом/выводом некритична. Они должны быть оснащены достаточно мощными блоками питания для возможности установки дополнительных плат расширения и дисковых накопителей. Желательно применение устройства бесперебойного питания. Оперативная память обычно имеет объем не менее 32 Мбайт, что позволит операционной системе (например, NetWare) использовать большие дисковые кэши и увеличить производительность сервера. Как правило, для работы с многозадачными операционными системами такие серверы оснащаются интерфейсом SCSI (или Fast SCSI). Распределение данных по нескольким жестким дискам может значительно повысить производительность.
При наличии одного сегмента сети и 10-20 рабочих станций пиковая пропускная способность сервера ограничивается максимальной пропускной способностью сети. В этом случае замена процессоров или дисковых подсистем более мощными не увеличивают производительность, так как узким местом является сама сеть. Поэтому важно использовать хорошую плату сетевого интерфейса.
Хотя влияние более быстрого процессора явно на производительности не сказывается, оно заметно снижает коэффициент использования ЦП. Во многих серверах этого класса используется процессоры Pentium с тактовой частотой 100 и более МГц, microSPARC-II и PowerPC. Аналогично процессорам влияние типа системной шины (EISA со скоростью 33 Мбит/с или PCI со скоростью 132 Мбит/с) также минимально при таком режиме использования.
Однако для файл-серверов общего доступа, с которыми одновременно могут работать несколько десятков, а то и сотен человек, простой однопроцессорной платформы и программного обеспечения Novell может оказаться недостаточно. В этом случае используются мощные многопроцессорные серверы с возможностями наращивания оперативной памяти до нескольких гигабайт, дискового пространства до сотен гигабайт, быстрыми интерфейсами дискового обмена (типа Fast SCSI-2, Fast&Wide SCSI-2 и Fiber Channel) и несколькими сетевыми интерфейсами. Эти серверы используют операционную систему UNIX, сетевые протоколы TCP/IP и NFS. На базе многопроцессорных UNIX-серверов обычно строятся также серверы баз данных крупных информационных систем, так как на них ложится основная нагрузка по обработке информационных запросов. Подобного рода серверы получили название суперсерверов.
По уровню общесистемной производительности, функциональным возможностям отдельных компонентов, отказоустойчивости, а также в поддержке многопроцессорной обработки, системного администрирования и дисковых массивов большой емкости суперсерверы вышли в настоящее время на один уровень с мейнфреймами и мощными миникомпьютерами. Современные суперсерверы характеризуются:

- • наличием двух или более центральных процессоров RISC, либо Pentium;
- • многоуровневой шинной архитектурой, в которой запатентованная высокоскоростная системная шина связывает между собой несколько процессоров и оперативную память, а также множество стандартных шин ввода/вывода, размещенных в том же корпусе;
- • поддержкой технологии дисковых массивов RAID;

- • поддержкой режима симметричной многопроцессорной обработки, которая позволяет распределять задания по нескольким центральным процессорам или режима асимметричной многопроцессорной обработки, которая допускает выделение процессоров для выполнения конкретных задач.

Как правило, суперсерверы работают под управлением операционных систем UNIX, а в последнее время и Windows NT, которые обеспечивают многопотоковую многопроцессорную и многозадачную обработку. Суперсерверы должны иметь достаточные возможности наращивания дискового пространства и вычислительной мощности, средства обеспечения надежности хранения данных и защиты от несанкционированного доступа. Кроме того, в условиях быстро растущей организации, важным условием является возможность наращивания и расширения уже существующей системы.

1.8. **2.4. Мейнфреймы**

Мейнфрейм - это синоним понятия "большая универсальная ЭВМ". Мейнфреймы и до сегодняшнего дня остаются наиболее мощными (не считая суперкомпьютеров) вычислительными системами общего назначения, обеспечивающими непрерывный круглосуточный режим эксплуатации. Они могут включать один или несколько процессоров, каждый из которых, в свою очередь, может оснащаться векторными сопроцессорами (ускорителями операций с суперкомпьютерной производительностью).

В нашем сознании мейнфреймы все еще ассоциируются с большими по габаритам машинами, требующими специально оборудованных помещений с системами водяного охлаждения и кондиционирования. Однако это не совсем так. Прогресс в области элементно-конструкторской базы позволил существенно сократить габариты основных устройств. Наряду со сверхмощными мейнфреймами, требующими организации двухконтурной водяной системы охлаждения, имеются менее мощные модели, для охлаждения которых достаточно принудительной воздушной вентиляции, и модели, построенные по блочно-модульному принципу и не требующие специальных помещений и кондиционеров.

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

В архитектурном плане мейнфреймы представляют собой многопроцессорные системы, содержащие один или несколько центральных и периферийных процессоров с общей памятью, связанных между собой высокоскоростными магистралями передачи данных. При этом основная вычислительная нагрузка ложится на центральные процессоры, а периферийные процессоры (в терминологии IBM - селекторные, блок-мультиплексные, мультиплексные каналы и процессоры телеобработки) обеспечивают работу с широкой номенклатурой периферийных устройств.

Первоначально мейнфреймы ориентировались на централизованную модель вычислений, работали под управлением патентованных операционных систем и имели ограниченные возможности для объединения в единую систему оборудования различных фирм-поставщиков. Однако повышенный интерес потребителей к открытым системам, построенным на базе международных стандартов и позволяющим достаточно эффективно использовать все преимущества такого подхода, заставил поставщиков мейнфреймов существенно расширить возможности своих операционных систем в направлении совместимости. В настоящее время они демонстрирует свою "открытость", обеспечивая соответствие со спецификациями POSIX 1003.3, возможность использования сетевых протоколов OSI и TCP/IP или предоставляя возможность работы на своих компьютерах под управлением операционной системы UNIX собственной разработки.

Стремительный рост производительности персональных компьютеров, рабочих станций и серверов создал тенденцию перехода с мейнфреймов на компьютеры менее дорогих классов: миникомпьютеры и многопроцессорные серверы. Эта тенденция получила название "разукрупнение" (downsizing). Однако этот процесс в самое последнее время несколько замедлился. Основной причиной возрождения интереса к мейнфреймам эксперты считают сложность перехода к распределенной архитектуре клиент-сервер, которая оказалась выше, чем предполагалось. Кроме того, многие пользователи считают, что распределенная среда не обладает достаточной надежностью для наиболее ответственных приложений, которой обладают мейнфреймы.

Очевидно выбор центральной машины (сервера) для построения информационной системы предприятия возможен только после глубокого анализа проблем, условий и требований конкретного заказчика и долгосрочного прогнозирования развития этой системы.

Главным недостатком мейнфреймов в настоящее время остается относительно низкое соотношение производительность/стоимость. Однако фирмами-поставщиками мейнфреймов предпринимаются значительные усилия по улучшению этого показателя.

Следует также помнить, что в мире существует огромная инсталлированная база мейнфреймов, на которой работают десятки тысяч прикладных программных систем. Отказаться от годами наработанного программного обеспечения просто не разумно. Поэтому в настоящее время ожидается рост продаж мейнфреймов по крайней мере до конца этого столетия. Эти системы, с одной стороны, позволят модернизировать существующие системы, обеспечив сокращение эксплуатационных расходов, с другой стороны, создадут новую базу для наиболее ответственных приложений.

1.9. **2.5. Системы высокой готовности**

Двумя основными проблемами построения вычислительных систем для критически важных приложений, связанных с обработкой транзакций, управлением базами данных и обслуживанием телекоммуникаций, являются обеспечение высокой производительности и продолжительного функционирования систем. Наиболее эффективный способ достижения заданного уровня производительности - применение параллельных масштабируемых архитектур. Задача обеспечения продолжительного функционирования системы имеет три составляющих: надежность, готовность и удобство обслуживания. Все эти три составляющих предполагают, в первую очередь, борьбу с неисправностями системы, порождаемыми отказами и сбоями в ее работе. Эта борьба ведется по всем трем направлениям, которые взаимосвязаны и применяются совместно.

Как уже отмечалось, повышение надежности основано на принципе предотвращения неисправностей путем снижения интенсивности отказов и сбоев за счет применения электронных схем и компонентов с высокой и сверхвысокой степенью интеграции, снижения уровня помех, облегченных режимов работы схем, обеспечение тепловых режимов их работы, а также за счет совершенствования методов сборки аппаратуры. Повышение уровня готовности предполагает подавление в определенных пределах влияния отказов и сбоев на работу системы с помощью средств контроля и коррекции ошибок, а также средств автоматического восстановления вычислительного процесса после проявления неисправности, включая аппаратурную и программную избыточность, на основе которой реализуются различные варианты отказоустойчивых архитектур. Повышение готовности есть способ борьбы за снижение времени простоя системы. Основные эксплуатационные характеристики системы существенно зависят от удобства ее обслуживания, в частности от ремонтопригодности, контролепригодности и т.д.

В последние годы в литературе по вычислительной технике все чаще употребляется термин "системы высокой готовности" (High Availability Systems). Все типы систем высокой готовности имеют общую цель - минимизацию времени простоя. Имеется два типа времени простоя компьютера: плановое и неплановое. Минимизация каждого из них требует различной стратегии и технологии. Плановое время простоя обычно включает время, принятое руководством, для проведения работ по модернизации системы и для ее обслуживания. Неплановое время простоя является результатом отказа системы или компонента. Хотя системы высокой готовности возможно больше ассоциируются с минимизацией неплановых простоев, они оказываются также полезными для уменьшения планового времени простоя.
Существует несколько типов систем высокой готовности, отличающиеся своими функциональными возможностями и стоимостью. Следует отметить, что высокая готовность не дается бесплатно. Стоимость систем высокой готовности намного превышает стоимость обычных систем. Вероятно поэтому наибольшее распространение в мире получили кластерные системы, благодаря тому, что они обеспечивают достаточно высокий уровень готовности систем при относительно низких затратах. Термин "кластеризация" на сегодня в компьютерной промышленности имеет много различных значений. Строгое определение могло бы звучать так: "реализация объединения машин, представляющегося единым целым для операционной системы, системного программного обеспечения, прикладных программ и пользователей". Машины, кластеризованные вместе таким способом могут при отказе одного процессора очень быстро перераспределить работу на другие процессоры внутри кластера. Это, возможно, наиболее важная задача многих поставщиков систем высокой готовности.
Первой концепцию кластерной системы анонсировала компания DEC, определив ее как группу объединенных между собой вычислительных машин, представляющих собой единый узел обработки информации. По существу VAX-кластер представляет собой слабосвязанную многомашинную систему с общей внешней памятью, обеспечивающую единый механизм управления и администрирования. В настоящее время на смену VAX-кластерам приходят UNIX-кластеры. При этом VAX-кластеры предлагают проверенный набор решений, который устанавливает критерии для оценки подобных систем.
VAX-кластер обладает следующими свойствами:
Разделение ресурсов. Компьютеры VAX в кластере могут разделять доступ к общим ленточным и дисковым накопителям. Все компьютеры VAX в кластере могут обращаться к отдельным файлам данных как к локальным.
Высокая готовность. Если происходит отказ одного из VAX-компьютеров, задания его пользователей автоматически могут быть перенесены на другой компьютер кластера. Если в системе имеется несколько контроллеров внешних накопителей и один из них отказывает, другие контроллеры автоматически подхватывают его работу.
Высокая пропускная способность. Ряд прикладных систем могут пользоваться возможностью параллельного выполнения заданий на нескольких компьютерах кластера.
Удобство обслуживания системы. Общие базы данных могут обслуживаться с единственного места. Прикладные программы могут инсталлироваться только однажды на общих дисках кластера и разделяться между всеми компьютерами кластера.
Расширяемость. Увеличение вычислительной мощности кластера достигается подключением к нему дополнительных VAX-компьютеров. Дополнительные накопители на магнитных дисках и магнитных лентах становятся доступными для всех компьютеров, входящих в кластер.
Работа любой кластерной системы определяется двумя главными компонентами: высокоскоростным механизмом связи процессоров между собой и системным программным обеспечением, которое обеспечивает клиентам прозрачный доступ к системному сервису.

В настоящее время широкое распространение получила также технология параллельных баз данных. Эта технология позволяет множеству процессоров разделять доступ к единственной базе данных. Распределение заданий по множеству процессорных ресурсов и параллельное их выполнение позволяет достичь более высокого уровня пропускной способности транзакций, поддерживать большее число одновременно работающих пользователей и ускорить выполнение сложных запросов. Существуют три различных типа архитектуры, которые поддерживают параллельные базы данных:

- • Симметричная многопроцессорная архитектура с общей памятью (Shared Memory SMP Architecture). Эта архитектура поддерживает единую базу данных, работающую на многопроцессорном сервере под управлением одной операционной системы. Увеличение производительности таких систем обеспечивается наращиванием числа процессоров, устройств оперативной и внешней памяти.
- • Архитектура с общими (разделяемыми) дисками (Shared Disk Architecture). Это типичный случай построения кластерной системы. Эта архитектура поддерживает единую базу данных при работе с несколькими компьютерами, объединенными в кластер (обычно такие компьютеры называются узлами кластера), каждый из которых работает под управлением своей копии операционной системы. В таких системах все узлы разделяют доступ к общим дискам, на которых собственно и располагается единая база данных. Производительность таких систем может увеличиваться как путем наращивания числа процессоров и объемов оперативной памяти в каждом узле кластера, так и посредством увеличения количества самих узлов.
- • Архитектура без разделения ресурсов (Shared Nothing Architecture). Как и в архитектуре с общими дисками, в этой архитектуре поддерживается единый образ базы данных при работе с несколькими компьютерами, работающими под управлением своих копий операционной системы. Однако в этой архитектуре каждый узел системы имеет собственную оперативную память и собственные диски, которые не разделяются между отдельными узлами системы. Практически в таких системах разделяется только общий коммуникационный канал между узлами системы. Производительность таких систем может увеличиваться путем добавления процессоров, объемов оперативной и внешней (дисковой) памяти в каждом узле, а также путем наращивания количества таких узлов.

Таким образом, среда для работы параллельной базы данных обладает двумя важными свойствами: высокой готовностью и высокой производительностью. В случае кластерной организации несколько компьютеров или узлов кластера работают с единой базой данных. В случае отказа одного из таких узлов, оставшиеся узлы могут взять на себя задания, выполнявшиеся на отказавшем узле, не останавливая общий процесс работы с базой данных. Поскольку логически в каждом узле системы имеется образ базы данных, доступ к базе данных будет обеспечиваться до тех пор, пока в системе имеется, по крайней мере, один исправный узел. Производительность системы легко масштабируется, т.е. добавление дополнительных процессоров, объемов оперативной и дисковой памяти, и новых узлов в системе может выполняться в любое время, когда это действительно требуется.

Параллельные базы данных находят широкое применение в системах обработки транзакций в режиме on-line, системах поддержки принятия решений и часто используются при работе с критически важными для работы предприятий и организаций приложениями, которые эксплуатируются по 24 часа в сутки.

2. 3. Методы оценки производительности

2.1. 3.1. Общие замечания

Основу для сравнения различных типов компьютеров между собой дают стандартные методики измерения производительности. В процессе развития вычислительной техники появилось несколько таких стандартных методик. Они позволяют разработчикам и пользователям осуществлять выбор между альтернативами на основе количественных показателей, что дает возможность постоянного прогресса в данной области.

Единицей измерения производительности компьютера является время: компьютер, выполняющий тот же объем работы за меньшее время является более быстрым. Время выполнения любой программы измеряется в секундах. Часто производительность измеряется как скорость появления некоторого числа событий в секунду, так что меньшее время подразумевает большую производительность.

Однако в зависимости от того, что мы считаем, время может быть определено различными способами. Наиболее простой способ определения времени называется астрономическим временем, временем ответа (response time), временем выполнения(execution time) или прошедшим временем (elapsed time). Это задержка выполнения задания, включающая буквально все: работу процессора, обращения к диску, обращения к памяти, ввод/вывод и накладные расходы операционной системы. Однако при работе в мультипрограммном режиме во время ожидания ввода/вывода для одной программы, процессор может выполнять другую программу, и система не обязательно будет минимизировать время выполнения данной конкретной программы.

Для измерения времени работы процессора на данной программе используется специальный параметр - время ЦП (CPU time), которое не включает время ожидания ввода/вывода или время выполнения другой программы. Очевидно, что время ответа, видимое пользователем, является полным временем выполнения программы, а не временем ЦП. Время ЦП может далее делиться на время, потраченное ЦП непосредственно на выполнение программы пользователя и называемое пользовательским временем ЦП, и время ЦП, затраченное операционной системой на выполнение заданий, затребованных программой, и называемое системным временем ЦП.

В ряде случаев системное время ЦП игнорируется из-за возможной неточности измерений, выполняемых самой операционной системой, а также из-за проблем, связанных со сравнением производительности машин с разными операционными системами. С другой стороны, системный код на некоторых машинах является пользовательским кодом на других и, кроме того, практически никакая программа не может работать без некоторой операционной системы. Поэтому при измерениях производительности процессора часто используется сумма пользовательского и системного времени ЦП.

В большинстве современных процессоров скорость протекания процессов взаимодействия внутренних функциональных устройств определяется не естественными задержками в этих устройствах, а задается единой системой синхросигналов, вырабатываемых некоторым генератором тактовых импульсов, как правило, работающим с постоянной скоростью. Дискретные временные события называются тактами синхронизации (clock ticks), просто тактами (ticks), периодами синхронизации (clock periods), циклами (cycles) или циклами синхронизации (clock cycles).Разработчики компьютеров обычно говорят о периоде синхронизации, который определяется либо своей длительностью (например, 10 наносекунд), либо частотой (например, 100 МГц). Длительность периода синхронизации есть величина, обратная к частоте синхронизации.

Таким образом, время ЦП для некоторой программы может быть выражено двумя способами: количеством тактов синхронизации для данной программы, умноженным на длительность такта синхронизации, либо количеством тактов синхронизации для данной программы, деленным на частоту синхронизации.

Важной характеристикой, часто публикуемой в отчетах по процессорам, является среднее количество тактов синхронизации на одну команду - CPI (clock cycles per instruction). При известном количестве выполняемых команд в программе этот параметр позволяет быстро оценить время ЦП для данной программы.

Таким образом, производительность ЦП зависит от трех параметров: такта (или частоты) синхронизации, среднего количества тактов на команду и количества выполняемых команд. Невозможно изменить ни один из указанных параметров изолированно от другого, поскольку базовые технологии, используемые для изменения каждого из этих параметров, взаимосвязаны: частота синхронизации определяется технологией аппаратных средств и функциональной организацией процессора; среднее количество тактов на команду зависит от функциональной организации и архитектуры системы команд; а количество выполняемых в программе команд определяется архитектурой системы команд и технологией компиляторов. Когда сравниваются две машины, необходимо рассматривать все три компоненты, чтобы понять относительную производительность.

В процессе поиска стандартной единицы измерения производительности компьютеров было принято несколько популярных единиц измерения, вследствие чего несколько безвредных терминов были искусственно вырваны из их хорошо определенного контекста и использованы там, для чего они никогда не предназначались. В действительности единственной подходящей и надежной единицей измерения производительности является время выполнения реальных программ, и все предлагаемые замены этого времени в качестве единицы измерения или замены реальных программ в качестве объектов измерения на синтетические программы только вводят в заблуждение.

Опасности некоторых популярных альтернативных единиц измерения (MIPS и MFLOPS) будут рассмотрены в соответствующих подразделах.

2.2. **3.2. MIPS**

Одной из альтернативных единиц измерения производительности процессора (по отношению к времени выполнения) является MIPS - (миллион команд в секунду). Имеется несколько различных вариантов интерпретации определения MIPS.

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

Положительными сторонами MIPS является то, что эту характеристику легко понять, особенно покупателю, и что более быстрая машина характеризуется большим числом MIPS, что соответствует нашим интуитивным представлениям. Однако использование MIPS в качестве метрики для сравнения наталкивается на три проблемы. Во-первых, MIPS зависит от набора команд процессора, что затрудняет сравнение по MIPS компьютеров, имеющих разные системы команд. Во-вторых, MIPS даже на одном и том же компьютере меняется от программы к программе. В-третьих, MIPS может меняться по отношению к производительности в противоположенную сторону.

Классическим примером для последнего случая является рейтинг MIPS для машины, в состав которой входит сопроцессор плавающей точки. Поскольку в общем случае на каждую команду с плавающей точкой требуется большее количество тактов синхронизации, чем на целочисленную команду, то программы, используя сопроцессор плавающей точки вместо соответствующих подпрограмм из состава программного обеспечения, выполняются за меньшее время, но имеют меньший рейтинг MIPS. При отсутствии сопроцессора операции над числами с плавающей точкой реализуются с помощью подпрограмм, использующих более простые команды целочисленной арифметики и, как следствие, такие машины имеют более высокий рейтинг MIPS, но выполняют настолько большее количество команд, что общее время выполнения значительно увеличивается. Подобные аномалии наблюдаются и при использовании оптимизирующих компиляторов, когда в результате оптимизации сокращается количество выполняемых в программе команд, рейтинг MIPS уменьшается, а производительность увеличивается.

Другое определение MIPS связано с очень популярным когда-то компьютером VAX 11/780 компании DEC. Именно этот компьютер был принят в качестве эталона для сравнения производительности различных машин. Считалось, что производительность VAX 11/780 равна 1MIPS (одному миллиону команд в секунду).

В то время широкое распространение получил синтетический тест Dhrystone, который позволял оценивать эффективность процессоров и компиляторов с языка C для программ нечисловой обработки. Он представлял собой тестовую смесь, 53% которой составляли операторы присваивания, 32% - операторы управления и 15% - вызовы функций. Это был очень короткий тест: общее число команд равнялось 100. Скорость выполнения программы из этих 100 команд измерялась в Dhrystone в секунду. Быстродействие VAX 11/780 на этом синтетическом тесте составляло 1757Dhrystone в секунду. Таким образом 1MIPS равен 1757 Dhrystone в секунду.

Следует отметить, что в настоящее время тест Dhrystone практически не применяется. Малый объем позволяет разместить все команды теста в кэш-памяти первого уровня современного микропроцессора и он не позволяет даже оценить эффект наличия кэш-памяти второго уровня, хотя может хорошо отражать эффект увеличения тактовой частоты.

Третье определение MIPS связано с IBM RS/6000 MIPS. Дело в том, что ряд производителей и пользователей (последователей фирмы IBM) предпочитают сравнивать производительность своих компьютеров с производительностью современных компьютеров IBM, а не со старой машиной компании DEC. Соотношение между VAX MIPS и RS/6000 MIPS никогда широко не публиковались, но 1 RS/6000 MIPS примерно равен 1.6 VAX 11/780 MIPS.

2.3. **3.3. MFLOPS**

Измерение производительности компьютеров при решении научно-технических задач, в которых существенно используется арифметика с плавающей точкой, всегда вызывало особый интерес. Именно для таких вычислений впервые встал вопрос об измерении производительности, а по достигнутым показателям часто делались выводы об общем уровне разработок компьютеров. Обычно для научно-технических задач производительность процессора оценивается в MFLOPS (миллионах чисел-результатов вычислений с плавающей точкой в секунду, или миллионах элементарных арифметических операций над числами с плавающей точкой, выполненных в секунду).

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

Ясно, что рейтинг MFLOPS зависит от машины и от программы. Этот термин менее безобидный, чем MIPS. Он базируется на количестве выполняемых операций, а не на количестве выполняемых команд. По мнению многих программистов, одна и та же программа, работающая на различных компьютерах, будет выполнять различное количество команд, но одно и то же количество операций с плавающей точкой. Именно поэтому рейтинг MFLOPS предназначался для справедливого сравнения различных машин между собой.

Однако и с MFLOPS не все обстоит так безоблачно. Прежде всего, это связано с тем, что наборы операций с плавающей точкой не совместимы на различных компьютерах. Например, в суперкомпьютерах фирмы Cray Research отсутствует команда деления (имеется, правда, операция вычисления обратной величины числа с плавающей точкой, а операция деления может быть реализована с помощью умножения делимого на обратную величину делителя). В то же время многие современные микропроцессоры имеют команды деления, вычисления квадратного корня, синуса и косинуса.

Другая, осознаваемая всеми, проблема заключается в том, что рейтинг MFLOPS меняется не только на смеси целочисленных операций и операций с плавающей точкой, но и на смеси быстрых и медленных операций с плавающей точкой. Например, программа со 100% операций сложения будет иметь более высокий рейтинг, чем программа со 100% операций деления.

Решение обеих проблем заключается в том, чтобы взять “каноническое” или "нормализованное" число операций с плавающей точкой из исходного текста программы и затем поделить его на время выполнения. На рисунке 2.1 показано, каким образом авторы тестового пакета "Ливерморские циклы", о котором речь пойдет ниже, вычисляют для программы количество нормализованных операций с плавающей точкой в соответствии с операциями, действительно находящимися в ее исходном тексте. Таким образом, рейтинг реальных MFLOPS отличается от рейтинга нормализованных MFLOPS, который часто приводится в литературе по суперкомпьютерам.

Реальные операции с ПТ	**Нормализованные операции с ПТ**
Сложение, вычитание, сравнение, умножение	1
Деление, квадратный корень	4
Экспонента, синус, ...	8

Рис. 3.1. Соотношение между реальными и нормализованными операциями с плавающей точкой, которым пользуются авторы “ливерморских циклов" для вычисления рейтинга MFLOPS

Наиболее часто MFLOPS, как единица измерения производительности, используется при проведении контрольных испытаний на тестовых пакетах "Ливерморские циклы" и LINPACK.

Ливерморские циклы - это набор фрагментов фортран-программ, каждый из которых взят из реальных программных систем, эксплуатируемых в Ливерморской национальной лаборатории им.Лоуренса (США). Обычно при проведении испытаний используется либо малый набор из 14 циклов, либо большой набор из 24 циклов.

Пакет Ливерморских циклов используется для оценки производительности вычислительных машин с середины 60-х годов. Ливерморские циклы считаются типичными фрагментами программ численных задач. Появление новых типов машин, в том числе векторных и параллельных, не уменьшило важности Ливерморских циклов,

однако изменились значения производительности и величины разброса между разными циклами.
На векторной машине производительность зависит не только от элементной базы, но и от характера самого алгоритма, т.е. коэффициента векторизуемости. Среди Ливерморских циклов коэффициент векторизуемости колеблется от 0 до 100%, что еще раз подтверждает их ценность для оценки производительности векторных архитектур. Кроме характера алгоритма, на коэффициент векторизуемости влияет и качество векторизатора, встроенного в компилятор.
На параллельной машине производительность существенно зависит от соответствия между структурой аппаратных связей вычислительных элементов и структурой вычислений в алгоритме. Важно, чтобы тестовый пакет представлял алгоритмы различных структур. В Ливерморских циклах встречаются последовательные, сеточные, конвейерные, волновые вычислительные алгоритмы, что подтверждает их пригодность и для параллельных машин. Однако обобщение результатов измерения производительности, полученных для одной параллельной машины, на другие параллельные машины или хотя бы некоторый подкласс параллельных машин, может дать неверный результат, ибо структуры аппаратных связей в таких машинах гораздо более разнообразны, чем, скажем, в векторных машинах.
LINPACK - это пакет фортран-программ для решения систем линейных алгебраических уравнений. Целью создания LINPACK отнюдь не было измерение производительности. Алгоритмы линейной алгебры весьма широко используются в самых разных задачах, и поэтому измерение производительности на LINPACK представляют интерес для многих пользователей. Сведения о производительности различных машин на пакете LINPACK публикуются сотрудником Аргоннской национальной лаборатории (США) Дж. Донгаррой и периодически обновляются.
В основе алгоритмов действующего варианта LINPACK лежит метод декомпозиции. Исходная матрица размером 100x100 элементов (в последнем варианте размером 1000x1000) сначала представляется в виде произведения двух матриц стандартной структуры, над которыми затем выполняется собственно алгоритм нахождения решения. Подпрограммы, входящие в LINPACK, структурированы. В стандартном варианте LINPACK выделен внутренний уровень базовых подпрограмм, каждая из которых выполняет элементарную операцию над векторами. Набор базовых подпрограмм называется BLAS (Basic Linear Algebra Subprograms). Например, в BLAS входят две простые подпрограммы SAXPY (умножение вектора на скаляр и сложение векторов) и SDOT (скалярное произведение векторов). Все операции выполняются над числами с плавающей точкой, представленными с двойной точностью. Результат измеряется в MFLOPS.
Использование результатов работы тестового пакета LINPACK с двойной точностью как основы для демонстрации рейтинга MFLOPS стало общепринятой практикой в компьютерной промышленности. При этом следует помнить, что при использовании исходной матрицы размером 100x100, она полностью может размещаться в кэш-памяти емкостью, например, 1 Мбайт. Если при проведении испытаний используется матрица размером 1000x1000, то емкости такого кэша уже недостаточно и некоторые обращения к памяти будут ускоряться благодаря наличию такого кэша, другие же будут приводить к промахам и потребуют большего времени на обработку обращений к памяти. Для многопроцессорных систем также имеются параллельные версии LINPACK, и такие системы часто показывают линейное увеличение производительности с ростом числа процессоров.
Однако, как и любая другая единица измерения, рейтинг MFLOPS для отдельной программы не может быть обобщен на все случаи жизни, чтобы представлять единственную единицу измерения производительности компьютера, хотя очень соблазнительно характеризовать машину единственным рейтингом MIPS или MFLOPS без указания программы.

2.4. **3.4. Тесты SPEC**

Важность создания пакетов тестов, базирующихся на реальных прикладных программах широкого круга пользователей и обеспечивающих эффективную оценку производительности процессоров, была осознана большинством крупнейших производителей компьютерного оборудования, которые в 1988 году учредили бесприбыльную корпорацию SPEC (Standard Performance Evaluation Corporation). Основной целью этой организации является разработка и поддержка стандартизованного набора специально подобранных тестовых программ для оценки производительности новейших поколений высокопроизводительных компьютеров. Членом SPEC может стать любая организация, уплатившая вступительный взнос.

Главными видами деятельности SPEC являются:

1. 1. Разработка и публикация наборов тестов, предназначенных для измерения производительности компьютеров. Перед публикацией объектные коды этих наборов вместе с исходными текстами и инструментальными средствами интенсивно проверяются на предмет возможности импортирования на разные платформы. Они доступны для широкого круга пользователей за плату, покрывающую расходы на разработку и административные издержки. Специальное лицензионное соглашение регулирует вопросы выполнения тестирования и публикации результатов в соответствии с документацией на каждый тестовый набор.
2. 2. SPEC публикует ежеквартальный отчет о новостях SPEC и результатах тестирования: “The SPEC Newsletter", что обеспечивает централизованный источник информации для результатов тестирования на тестах SPEC.

Основным результатом работы SPEC являются наборы тестов. Эти наборы разрабатываются SPEC с использованием кодов, поступающих из разных источников. SPEC работает над импортированием этих кодов на разные платформы, а также создает инструментальные средства для формирования из кодов, выбранных в качестве тестов, осмысленных рабочих нагрузок. Поэтому тесты SPEC отличаются от свободно распространяемых программ. Хотя они могут существовать под похожими или теми же самыми именами, время их выполнения в общем случае будет отличаться.

В настоящее время имеется два базовых набора тестов SPEC, ориентированных на интенсивные расчеты и измеряющих производительность процессора, системы памяти, а также эффективность генерации кода компилятором. Как правило, эти тесты ориентированы на операционную систему UNIX, но они также импортированы и на другие платформы. Процент времени, расходуемого на работу операционной системы и функции ввода/вывода, в общем случае ничтожно мал.

Набор тестов CINT92, измеряющий производительность процессора при обработке целых чисел, состоит из шести программ, написанных на языке Си и выбранных из различных прикладных областей: теория цепей, интерпретатор языка Лисп, разработка логических схем, упаковка текстовых файлов, электронные таблицы и компиляция программ.

Набор тестов CFP92, измеряющий производительность процессора при обработке чисел с плавающей точкой, состоит из 14 программ, также выбранных из различных прикладных областей: разработка аналоговых схем, моделирование методом Монте-Карло, квантовая химия, оптика, робототехника, квантовая физика, астрофизика, прогноз погоды и другие научные и инженерные задачи. Две программы из этого набора написаны на языке Си, а остальные 12 - на Фортране. В пяти программах используется одинарная, а в остальных - двойная точность.

Результаты прогона каждого индивидуального теста из этих двух наборов выражаются отношением времени выполнения одной копии теста на тестируемой машине к времени ее выполнения на эталонной машине. В качестве эталонной машины используется VAX 11/780. SPEC публикует результаты прогона каждого отдельного теста, а также две составные оценки: SPECint92 - среднее геометрическое 6 результатов индивидуальных тестов из набора CINT92 и SPECfp92 - среднее геометрическое 14 результатов индивидуальных тестов из набора CFP92.

Следует отметить, что результаты тестирования на наборах CINT92 и CFT92 сильно зависят от качества применяемых оптимизирующих компиляторов. Для более точного выяснения возможностей аппаратных средств с середины 1994 года SPEC ввел две дополнительные составные оценки: SPECbase_int92 и SPECbase_fp92, которые накладывает определенные ограничения на используемые компиляторы поставщиками компьютеров при проведении испытаний.

Составные оценки SPECint92 и SPECfp92 достаточно хорошо характеризуют производительность процессора и системы памяти при работе в однозадачном режиме, но они совершенно не подходят для оценки производительности многопроцессорных и однопроцессорных систем, работающих в многозадачном режиме. Для этого нужна оценка пропускной способности системы или ее емкости, показывающая количество заданий, которое система может выполнить в течение заданного интервала времени. Пропускная способность системы определяется прежде всего количеством ресурсов (числом процессоров, емкостью оперативной и кэш-памяти, пропускной способностью шины), которые система может предоставить в распоряжение пользователя в каждый момент времени. Именно такую оценку, названную SPECrate и заменившую ранее применявшуюся оценку SPECthruput89, SPEC предложила в качестве единицы измерения производительности многопроцессорных систем.

При этом для измерения выбран метод "однородной нагрузки" (homogenous capacity metod), заключающийся в том, что одновременно выполняются несколько копий одной и той же тестовой программы. Результаты этих тестов показывают, как много задач конкретного типа могут быть выполнены в указанное время, а их средние геометрические значения (SPECrate_int92 - на наборе тестов, измеряющих производительность целочисленных операций и SPECrate_fp92 - на наборе тестов, измеряющих производительность на операциях с плавающей точкой) наглядно отражают пропускную способность однопроцессорных и многопроцессорных конфигураций при работе в многозадачном режиме в системах коллективного пользования. В качестве тестовых программ для проведения испытаний на пропускную способность выбраны те же наборы CINT92 и CFT92.

При прогоне тестового пакета делаются независимые измерения по каждому отдельному тесту. Обычно такой параметр, как количество запускаемых копий каждого отдельного теста, выбирается исходя из соображений оптимального использования ресурсов, что зависит от архитектурных особенностей конкретной системы. Одной из очевидных возможностей является установка этого параметра равным количеству процессоров в системе. При этом все копии отдельной тестовой программы запускаются одновременно, и фиксируется время завершения последней из всех запущенных программ.

С середины 1994 года SPEC ввела две дополнительные составные оценки: SPECrate_base_int92 и SPECrate_base_fp92, которые накладывает ограничения на используемые компиляторы.

Следует отметить, что SPEC объявила о полном переходе с середины 1996 года на новый (третий) комплект тестов - CINT95, CFP95. Эти тесты удовлетворяют следующим ограничениям и требованиям:

- • размер кода и данных должен быть достаточно большим, чтобы он гарантированно не размещался целиком в кэш-памяти
- • время выполнения тестов должно быть увеличено с секунд до минут
- • используемые фрагменты программ должны быть реалистичными
- • применение усовершенствованного способа измерения времени

- • реализация более удобных инструментальных средств
- • стандартизация требований к компиляторам и методов вызова

Новый комплект тестов состоит из 8 целочисленных программ, написанных на языке Си и 10 программ вещественной арифметики, написанных на Фортране. Новые метрики получили соответствующие названия: SPECint95, SPECfp95, SPECint_base95, SPECfp_base95, SPECrate_int95, SPECrate_fp95, SPECrate_base_int95 и SPECrate_base_fp95.

2.5. **3.5. Тесты TPC**

По мере расширения использования компьютеров при обработке транзакций в сфере бизнеса все более важной становится возможность справедливого сравнения систем между собой. С этой целью в 1988 году был создан Совет по оценке производительности обработки транзакций (TPC - Transaction Processing Performance Council), который представляет собой station беcприбыльную организацию. Любая компания или организация может стать членом TPC после уплаты соответствующего взноса. На сегодня членами TPC являются практически все крупнейшие производители аппаратных платформ и программного обеспечения для автоматизации коммерческой деятельности. К настоящему времени TPC создал три тестовых пакета для обеспечения объективного сравнения различных систем обработки транзакций и планирует создать новые оценочные тесты.

В компьютерной индустрии термин транзакция (transaction) может означать почти любой вид взаимодействия или обмена информацией. Однако в мире бизнеса "транзакция" имеет вполне определенный смысл: коммерческий обмен товарами, услугами или деньгами. В настоящее время практически все бизнес-транзакции выполняются с помощью компьютеров. Наиболее характерными примерами систем обработки транзакций являются системы управления учетом, системы резервирования авиабилетов и банковские системы. Таким образом, необходимость стандартов и тестовых пакетов для оценки таких систем все больше усиливается. До 1988 года отсутствовало общее согласие относительно методики оценки систем обработки транзакций. Широко использовались два тестовых пакета: Дебет/Кредит и TPI. Однако эти пакеты не позволяли осуществлять адекватную оценку систем: они не имели полных, основательных спецификаций; не давали объективных, проверяемых результатов; не содержали полного описания конфигурации системы, ее стоимости и методологии тестирования; не обеспечивали объективного, беспристрастного сравнения одной системы с другой.

Чтобы решить эти проблемы, и была создана организация TPC, основной задачей которой является точное определение тестовых пакетов для оценки систем обработки транзакций и баз данных, а также для распространения объективных, проверяемых данных в промышленности.

TPC публикует спецификации тестовых пакетов, которые регулируют вопросы, связанные с работой тестов. Эти спецификации гарантируют, что покупатели имеют объективные значения данных для сравнения производительности различных вычислительных систем. Хотя реализация спецификаций оценочных тестов оставлена на усмотрение индивидуальных спонсоров тестов, сами спонсоры, объявляя результаты TPC, должны представить TPC детальные отчеты, документирующие соответствие всем спецификациям. Эти отчеты, в частности, включают конфигурацию системы, методику калькуляции цены, диаграммы значений производительности и документацию, показывающую, что тест соответствует требованиям атомарности, согласованности, изолированности и долговечности (ACID - atomicity, consistency, isolation, and durability), которые гарантируют, что все транзакции из оценочного теста обрабатываются должным образом. Обычно при описании конфигурации системы приводятся блок-схемы подключения каналов и устройств ввода/вывода. Детальный список аппаратных средств и программного обеспечения включает номера составных частей, их описание и номер версии/ревизии; в цену системы включена стоимость аппаратных средств и программного обеспечения, а также стоимость запасных частей, необходимых для эксплуатации системы в течение 5 лет; объем внешней памяти системы должен обеспечивать хранение информации о транзакциях за период 30, 90 или 180 дней.

Работой TPC руководит Совет Полного Состава (Full Council), который принимает все решения; каждая компания-участник имеет один голос, а для того, чтобы провести какое-либо решение, требуется две трети голосов. Управляющий Комитет (Steering Committee), состоящий из пяти представителей и избираемый ежегодно, надзирает за работой администрации TPC, поддерживает и обеспечивает все направления и рекомендации для членов Совета Полного Состава и Управляющего Комитета. В составе TPC имеются два типа подкомитетов: постоянные подкомитеты, которые управляют администрацией TPC, осуществляют связи с общественностью и обеспечивают выпуск документации; и технические подкомитеты, которые формируются для разработки предложений по оценочным тестам и распускаются после того, как их работа по разработке завершена.

Возможно наиболее важным аспектом тестов TPC является требование полного раскрытия всех деталей проведения испытаний. Информация, содержащаяся в отчете о проведении испытаний (FDR - Full Disclosure Report), должна обеспечивать возможность полного воспроизведения результатов. TPC также следит за тем, чтобы до появления FDR никакая информация не публиковалась. Каждый отчет проверяется советом технических советников, который состоит из ведущих специалистов по базам данных.

Тесты TPC

TPC определяет и управляет форматом нескольких тестов для оценки производительности OLTP (On-Line Transaction Processing), включая тесты TPC-A, TPC-B, TPC-C, TPC-D и TPC-E. Как уже отмечалось, создание оценочного теста является ответственностью организации, выполняющей этот тест. TPC требует только, чтобы при создании оценочного теста выполнялись определенные условия. Хотя упомянутые тесты TPC не представляют собой тесты для непосредственной оценки производительности баз данных, системы реляционных баз данных являются ключевыми компонентами любой системы обработки транзакций.

Следует отметить, что, как и любой другой тест, ни один тест TPC не может измерить производительность системы, которая применима для любой возможной среды обработки транзакций, но эти тесты действительно могут помочь пользователю справедливо сравнивать похожие системы. Однако, когда пользователь делает покупку или планирует решение о покупке, он должен понимать, что никакой тест не может заменить его конкретную прикладную задачу.

Тест TPC-A

Выпущенный в ноябре 1989 года, тест TCP-A предназначался для оценки производительности систем, работающих в среде интенсивно обновляемых баз данных, типичной для приложений интерактивной обработки данных (OLDP - on-line data processing). Такая среда характеризуется:

- • множеством терминальных сессий в режиме on-line
- • значительным объемом ввода/вывода при работе с дисками
- • умеренным временем работы системы и приложений
- • целостностью транзакций.

Практически, при выполнении теста, эмулируется типичная вычислительная среда банка, включающая сервер базы данных, терминалы и линии связи. Этот тест использует одиночные, простые транзакции, интенсивно обновляющие базу данных. Одиночная транзакция (подобная обычной операции обновления счета клиента) обеспечивает простую, повторяемую единицу работы, которая проверяет ключевые компоненты системы OLTP. Более подробно транзакция состоит из выполнения следующих действий:

- • обновление счета клиента (дебет/кредит)
- • обновление суммы наличных денег у кассира (дебет/кредит)
- • обновление общей суммы наличных денег в филиале банка (дебет/кредит)
- • запись номера счета клиента, филиала, кассира, суммы и даты операции в файл истории.

Тест TPC-A определяет пропускную способность системы, измеряемую количеством транзакций в секунду (tps A), которые система может выполнить при работе с множеством терминалов. Одной из неопределенностей старого теста Дебет/Кредит, которая часто использовалась поставщиками систем, была возможность подгонки соотношений между объемами таблиц СЧЕТ/ФИЛИАЛ/КАССИР, позволяющая обойти узкие места в подсистемах ввода/вывода и блокировок. В тесте TPC-A (а также в тесте TPC-B) соотношение межу количеством строк в таблицах СЧЕТ, ФИЛИАЛ и КАССИР строго специфицировано и для каждого сообщаемого в отчете уровня tpsA (tpsB) размер базы данных как минимум должен быть следующим:

СЧЕТ	100000 * tpsmin
КАССИР	10 * tpsmin
ФИЛИАЛ	1 * tpsmin
Количество терминалов	10 * tpsmin,

где tpsmin должно быть больше, чем приводимый в отчете рейтинг системы. Таким образом, для системы, выполняющей 2000 транзакций в секунду, таблица СЧЕТ должна содержать 200 миллионов записей и т.д. Кроме того, должно быть гарантировано, что представленная в отчете скорость транзакций на специфицированной системе должна устойчиво поддерживаться минимально в течение 8-часового периода непрерывной работы системы (хотя реально измерения на тесте могли проводиться и в течение только одного часа). Тест TPC-A требует наличия внешней памяти для хранения информации об истории всех транзакций, накапливающейся в течение 90 дней в предположении о сохранении установившейся скорости работы в течение 8-часового рабочего дня.

В тесте TPC-A специфицировано также "время обдумывания" пользователя, которое должно составлять, по крайней мере, 10 секунд на терминал. Это означает, что никакой терминал не может выдавать транзакции со скоростью более 0.1 транзакции в секунду. Таким образом, в состав конфигурации системы, обеспечивающей рейтинг в 2000 tpsA, должно входить, по крайней мере, 20000 терминалов, последовательных портов и других аппаратных средств поддержки межсоединений. Чтобы обеспечить подобные требования к системе межсоединений, необходимо соответствующее количество рабочих станций, концентраторов или мультиплексоров и в отчете должна быть приведена блок-схема всей системы межсоединений. Правда, в реальной жизни при проведении испытаний, как правило, терминалы эмулируются с помощью специальных эмуляторов удаленных терминалов (RTE - Remote Terminal Emulator) через соответствующие последовательные порты.

Тест TPC-A может выполняться в локальных или региональных вычислительных сетях. В этом случае его результаты определяют либо "локальную" пропускную способность (TPC-A-local Throughput), либо "региональную" пропускную способность (TPC-A-wide Throughput). Очевидно, эти два тестовых показателя нельзя непосредственно сравнивать. Спецификация теста TPC-A требует, чтобы все компании полностью раскрывали детали работы своего теста, свою конфигурацию системы и ее стоимость (с учетом пятилетнего срока обслуживания). Это позволяет определить нормализованную стоимость системы ($/tpsA).

Тест TPC-B

В августе 1990 года TPC одобрил TPC-B, интенсивный тест базы данных, характеризующийся следующими элементами:

- • значительный объем дискового ввода/вывода;
- • умеренное время работы системы и приложений;
- • целостность транзакций.

TPC-B измеряет пропускную способность системы в транзакциях в секунду (tpsB). Поскольку имеются существенные различия между двумя тестами TPC-A и TPC-B (в частности, в TPC-B не выполняется эмуляция терминалов и линий связи), их нельзя прямо сравнивать. На рис. 3.2 показаны взаимоотношения между TPC-A и TPC-B.

Тест TPC-C

Тестовый пакет TPC-C с точки зрения реальных потребностей потребителей сделал огромный шаг вперед по отношению к тестам TPC-A и TPC-B. Хотя по своей сути он также моделирует оперативную обработку транзакций, его сложность по крайней мере на порядок превышает сложность тестов A и B: он использует несколько типов транзакций, более сложную базу данных и общую структуру выполнения. Тест TPC-C моделирует прикладную задачу обработки заказов. Он моделирует достаточно сложную систему OLTP, которая должна управлять приемом заказов, управлением учетом товаров и распространением товаров и услуг. Тест TPC-C осуществляет тестирование всех основных компонентов системы: терминалов, линий связи, ЦП, дискового в/в и базы данных. TPC-C специфицирует время обдумывания и ввода с клавиатуры, которые обычно программируются в RTU при проведении испытаний.

TPC-C требует, чтобы выполнялись пять типов транзакций:

- • новый заказ, вводимый с помощью сложной экранной формы;
- • простое обновление базы данных, связанное с платежом;
- • простое обновление базы данных, связанное с поставкой;
- • справка о состоянии заказов;
- • справка по учету товаров.

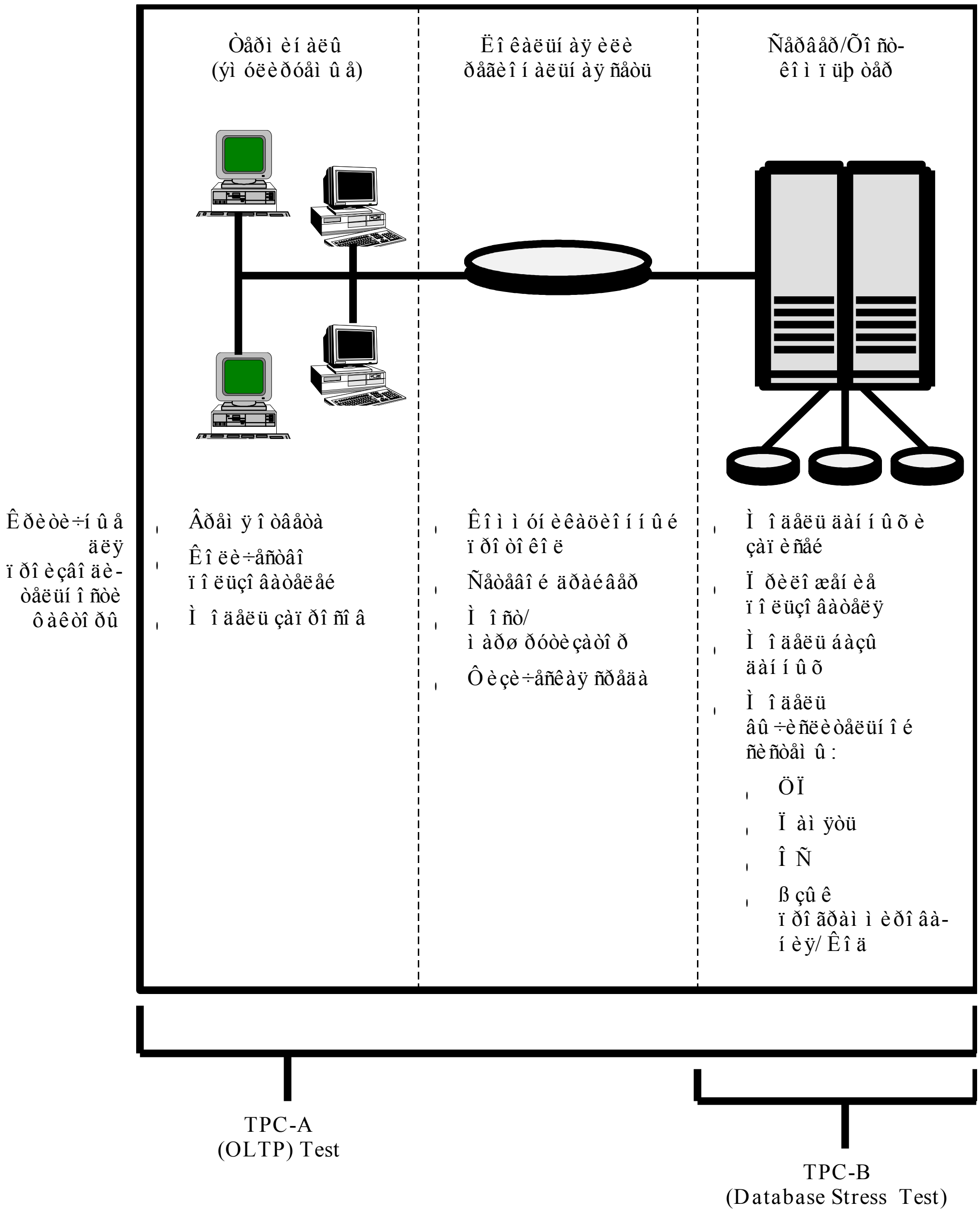


Рис. 3.2. Типовая среда обработки транзакций и соответствующие оценочные тесты TPC

Среди этих пяти типов транзакций, по крайней мере, 43% должны составлять платежи. Транзакции, связанные со справками о состоянии заказов, состоянии поставки и учета, должны составлять по 4%. Тест измеряет скорость транзакций по новым заказам, обрабатываемых совместно со смесью других транзакций, выполняющихся в фоновом режиме.

База данных TPC-C основана на модели оптового поставщика с удаленными районами и товарными складами. База данных содержит девять таблиц: товарные склады, район, покупатель, заказ, порядок заказов, новый заказ, статья счета, складские запасы и история.

Обычно публикуются два результата (таблица 3.2). Один из них, tpmC, представляет пиковую скорость выполнения транзакций (выражается в количестве транзакций в минуту и представляет собой максимальную пропускную способность системы (MQTh - Maximum Qualified Throghput)). Второй результат, $/tpmC, представляет собой нормализованную стоимость системы. Стоимость системы включает все аппаратные средства и программное обеспечение, используемые в тесте, плюс стоимость обслуживания в течение пяти лет.

Таблица 3.2.

TPC-C Results				
Company	System	Throughput (tpmC)	Price/Perf ($/tpmC)	Database Software
Compaq	ProLiant 5000 6/166 4/Pentium Pro/166MHz	6184.90	$111	Microsoft SQL Server 6.5
Compaq	ProLiant 5000 6/200 4/Pentium Pro/200MHz	6750.53	$90	Microsoft SQL Server 6.5
Digital	AlphaServer 8400 5/350 8/DECchip21164/350MHz	14227.25	$269	Oracle Rdb7 V 7.0
Digital	AlphaServer 4100 5/400 4/DECchip21164/400MHz	7985.15	$174	Oracle Rdb7 V 7.0
Digital	AlphaServer 4100 5/400 4/DECchip21164/400MHz	7598.63	$152	Sybase SQL Server 11.0
HP	HP 9000 Model D370 2/PA-RISC 8000/160MHz	5822.23	$148	Sybase SQL Server 11.0.3
HP	HP 9000 Model K460 4/PA-RISC 8000/180MHz	12321.87	$187	Sybase SQL Server 11.0.3
IBM	RS6000 PowerPC Server J40 8/Power PC 604/112MHz	5774.07	$243	Sybase SQL Server 11.0.3
SGI	Challenge XL Server 16/R4400/250MHz	6313.78	$479	Informix OnLine V.7.11.UDI
Sun	Ultra Enterprise 4000 12/UltraSPARC/167 MHz	11465.93	$189	Sybase SQL Server 11.0.2
Sun	Ultra Enterprise 3000 6/UltraSPARC/167 MHz	6662.47	$152	Sybase SQL Server 11.0.2

Масштаб системы расширяется путем увеличения количества товарных складов, причем каждый товарный склад должен поддерживать:

- • максимально 11.5 tpmC MQTh (приводимая в отчете метрика);
- • примерно 26 транзакций различной сложности в минуту;
- • 10 терминалов со средним временем обдумывания и ввода данных равным 23 секунды;
- • 367 Мбайт (неформатированных) данных для хранения истории за период в 180 дней.

Таким образом, рейтинг в 10000 tpmC MQTh предполагает примерно 0.5 терабайт внешней памяти!

Будущие тесты TCP
Сравнительно недавно TPC объявил об отмене тестов TPC-A и TPC-B. Отныне для оценки систем будут применяться существующий тестовый пакет TPC-C, новые тесты TPC-D и TPC-E, а также два еще полностью не разработанных теста. Представленный в первом квартале 1995 года тест TPC-D предназначен для оценки производительности систем принятия решений. Для оценки систем масштаба предприятия во втором квартале 1995 года TPC должен был представить тест TPC-E и его альтернативный вариант, не имеющий пока названия. Кроме того, TPC продолжает разработку тестовых пакетов для оценки баз данных и систем клиент/сервер. Первые результаты, полученные с помощью этих новых методов, уже начали публиковаться.

2.6. **3.6. Тесты AIM**

Одной из независимых организаций, осуществляющей оценку производительности вычислительных систем, является частная компания AIM Technology, которая была основана в 1981 году. Компания разрабатывает и поставляет программное обеспечение для измерения производительности систем, а также оказывает услуги по тестированию систем конечным пользователям и поставщикам вычислительных систем и сетей, которые используют промышленные стандартные операционные системы, такие как UNIX и OS/2.
За время своего существования компания разработала специальное программное обеспечение, позволяющее легко создавать различные рабочие нагрузки, соответствующие уровню тестируемой системы и требованиям по ее использованию. Это программное обеспечение состоит из двух основных частей: генератора тестовых пакетов (Benchmark Generator) и нагрузочных смесей (Load Mixes) прикладных задач.
Генератор тестовых пакетов представляет собой программную систему, которая обеспечивает одновременное выполнение множества программ. Он содержит большое число отдельных тестов, которые потребляют определенные ресурсы системы, и тем самым акцентируют внимание на определенных компонентах, из которых складывается ее общая производительность. При каждом запуске генератора могут выполняться любые отдельные или все доступные тесты в любом порядке и при любом количестве проходов, позволяя тем самым создавать для системы практически любую необходимую рабочую нагрузку. Все это дает возможность тестовому пакету моделировать любой тип смеси при постоянной смене акцентов (для лучшего представления реальной окружающей обстановки) и при обеспечении высокой степени конфигурирования.
Каждая нагрузочная смесь представляют собой формулу, которая определяет компоненты требуемой нагрузки. Эта формула задается в терминах количества различных доступных тестов, которые должны выполняться одновременно для моделирования рабочей нагрузки.
Используя эти две части программного обеспечения AIM, можно действительно создать для тестируемой системы любую рабочую нагрузку, определяя компоненты нагрузки в терминах тестов, которые должны выполняться генератором тестовых пакетов. Если некоторые требуемые тесты отсутствуют в составе генератора тестовых пакетов, то они могут быть легко туда добавлены.
Генератор тестовых пакетов во время своей работы "масштабирует" или увеличивает нагрузку на систему. Первоначально он выполняет и хронометрирует одну копию нагрузочной смеси. Затем одновременно выполняет и хронометрирует три копии нагрузочной смеси и т.д. По мере увеличения нагрузки, на основе оценки производительности системы, выбираются различные уровни увеличения нагрузки. В конце концов, может быть нарисована кривая пропускной способности, показывающая

возможности системы по обработке нагрузочной смеси в зависимости от числа моделируемых нагрузок. Это позволяет с достаточной достоверностью дать заключение о возможностях работы системы при данной нагрузке или при изменении нагрузки.

Очевидно, что сам по себе процесс моделирования рабочей нагрузки мало что дал бы для сравнения различных машин между собой при отсутствии у AIM набора хорошо подобранных смесей, которые представляют собой ряд важных для пользователя прикладных задач.

Все смеси AIM могут быть разделены на две категории: стандартные и заказные. Заказные смеси создаются для точного моделирования особенностей среды конечного пользователя или поставщика оборудования. Заказная смесь может быть тесно связана с определенными тестами, добавляемыми к генератору тестовых пакетов. В качестве альтернативы заказная смесь может быть связана с очень специфическим приложением, которое создает для системы необычную нагрузку. В общем случае заказные смеси разрабатываются на основе одной из стандартных смесей AIM путем ее "подгонки" для более точного представления определенной ситуации. Обычно заказные смеси разрабатываются заказчиком совместно с AIM Technology, что позволяет использовать многолетний опыт AIM по созданию и моделированию нагрузочных смесей.

К настоящему времени AIM создала восемь стандартных смесей, которые представляют собой обычную среду прикладных задач. В состав этих стандартных смесей входят:

1. 1. Универсальная смесь для рабочих станций (General Workstation Mix) - моделирует работу рабочей станции в среде разработки программного обеспечения.

2. 2. Смесь для механического САПР (Mechanical CAD Mix) моделирует рабочую станцию, используемую для трехмерного моделирования и среды системы автоматизации проектирования в механике.

3. 3. Смесь для геоинформационных систем (GIS Mix) - моделирует рабочую станцию, используемую для обработки изображений и в приложениях геоинформацинных систем.

4. 4. Смесь универсальных деловых приложений (General Business) - моделирует рабочую станцию, используемую для выполнения таких стандартных инструментальных средств, как электронная почта, электронные таблицы, база данных, текстовый процессор и т.д.

5. 5. Многопользовательская смесь (Shared/Multiuser Mix) моделирует многопользовательскую систему, обеспечивающую обслуживание приложений для множества работающих в ней пользователей.

6. 6. Смесь для вычислительного (счетного) сервера (ComputeServer Mix) - моделирует систему, используемую для выполнения заданий с большим объемом вычислений, таких как маршрутизация PCB, гидростатическое моделирование, вычислительная химия, взламывание кодов и т.д.

7. 7. Смесь для файл-сервера (File Server Mix) - моделирует запросы, поступающие в систему, используемую в качестве централизованного файлового сервера, включая ввод/вывод и вычислительные мощности для других услуг по запросу.

8. 8. Смесь СУБД (RBMS Mix) - моделирует систему, выполняющую ответственные приложения управления базой данных.

Одним из видов деятельность AIM Technology является выпуск сертифицированных отчетов по результатам тестирования различных систем. В качестве примера рассмотрим форму отчета AIM Performance Report II - независимое сертифицированное заключение о производительности системы.

Ключевыми частями этого отчета являются:

- стоимость системы,
- детали конфигурации системы,
- результаты измерения производительности, показанные на трех тестовых пакетах AIM.

Используются следующие три тестовых пакета:

- многопользовательский тестовый пакет AIM (набор III),
- тестовый пакет утилит AIM (Milestone),
- тестовый пакет для оценки различных подсистем (набор II).

В частности, набор III, разработанный компанией AIM Technology, используется в различных формах уже более 10 лет. Он представляет собой пакет тестов для системы UNIX, который пытается оценить все аспекты производительности системы, включая все основные аппаратные средства, используемые в многопрограммной среде. Этот тестовый пакет моделирует многопользовательскую работу в среде разделения времени путем генерации возрастающих уровней нагрузки на ЦП, подсистему ввода/вывода, переключение контекста и измеряет производительность системы при работе с множеством процессов.

Для оценки и сравнения систем в AIM Performance Report II используются следующие критерии:

- Пиковая производительность (рейтинг производительности по AIM)
- Максимальная пользовательская нагрузка
- Индекс производительности утилит
- Пропускная способность системы

Рейтинг производительности по AIM - стандартная единица измерения пиковой производительности, установленная AIM Technology. Этот рейтинг определяет наивысший уровень производительности системы, который достигается при оптимальном использовании ЦП, операций с плавающей точкой и кэширования диска. Рейтинг вездесущей машины VAX 11/780 обычно составляет 1 AIM. В отчетах AIM представлен широкий ряд UNIX-систем, которые можно сравнивать по этому параметру.

Максимальная пользовательская нагрузка - определяет "емкость" (capacity) системы, т.е. такую точку, начиная с которой производительность системы падает ниже приемлемого уровня для N-го пользователя (меньше чем одно задание в минуту на одного пользователя).

Индекс производительности утилит - определяет количество пользовательских нагрузок пакета Milestone, которые данная система выполняет в течение одного часа. Набор тестов Milestone многократно выполняет выбранные утилиты UNIX в качестве основных и фоновых заданий при умеренных пользовательских нагрузках. Этот параметр показывает возможности системы по выполнению универсальных утилит UNIX.

Максимальная пропускная способность - определяет пиковую производительность мультипрограммной системы, измеряемую количеством выполненных заданий в минуту. Приводящийся в отчете график пропускной способности системы показывает, как она работает при различных нагрузках.

Отчет по производительности разработан с использованием набора тестов AIM собственной разработки. В отличие от многих популярных тестовых пакетов, которые измеряют только производительность ЦП в однозадачном режиме и/или на операциях с плавающей точкой, тестовые пакеты AIM проверяют итоговую производительность системы и всех ее основных компонентов в многозадачной среде, включая ЦП, плавающую точку, память, диски, системные и библиотечные вызовы.

Синтетические ядра и натуральные тесты не могут служить в качестве настоящих тестовых пакетов для оценки систем: они не могут моделировать точно среду конечного пользователя и оценивать производительность всех относящихся к делу компонентов системы. Без такой гарантии результаты измерения производительности остаются под вопросом.

3. 4. Основные архитектурные понятия

3.1. 4.1. Архитектура системы команд. Классификация процессоров (CISC и RISC)

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

Двумя основными архитектурами набора команд, используемыми компьютерной промышленностью на современном этапе развития вычислительной техники являются архитектуры CISC и RISC. Основоположником CISC-архитектуры можно считать компанию IBM с ее базовой архитектурой /360, ядро которой используется с1964 года и дошло до наших дней, например, в таких современных мейнфреймах как IBM ES/9000.
Лидером в разработке микропроцессоров с полным набором команд (CISC - Complete Instruction Set Computer) считается компания Intel со своей серией x86 и Pentium. Эта архитектура является практическим стандартом для рынка микрокомпьютеров. Для CISC-процессоров характерно: сравнительно небольшое число регистров общего назначения; большое количество машинных команд, некоторые из которых нагружены семантически аналогично операторам высокоуровневых языков программирования и выполняются за много тактов; большое количество методов адресации; большое количество форматов команд различной разрядности; преобладание двухадресного формата команд; наличие команд обработки типа регистр-память.
Основой архитектуры современных рабочих станций и серверов является архитектура компьютера с сокращенным набором команд (RISC - Reduced Instruction Set Computer). Зачатки этой архитектуры уходят своими корнями к компьютерам CDC6600, разработчики которых (Торнтон, Крэй и др.) осознали важность упрощения набора команд для построения быстрых вычислительных машин. Эту традицию упрощения архитектуры С. Крэй с успехом применил при создании широко известной серии суперкомпьютеров компании Cray Research. Однако окончательно понятие RISC в современном его понимании сформировалось на базе трех исследовательских проектов компьютеров: процессора 801 компании IBM, процессора RISC университета Беркли и процессора MIPS Стенфордского университета.
Разработка экспериментального проекта компании IBM началась еще в конце 70-х годов, но его результаты никогда не публиковались, и компьютер на его основе в промышленных масштабах не изготавливался. В 1980 году Д.Паттерсон со своими коллегами из Беркли начали свой проект и изготовили две машины, которые получили названия RISC-I и RISC-II. Главными идеями этих машин было отделение медленной памяти от высокоскоростных регистров и использование регистровых окон. В 1981году

Дж.Хеннесси со своими коллегами опубликовал описание стенфордской машины MIPS, основным аспектом разработки которой была эффективная реализация конвейерной обработки посредством тщательного планирования компилятором его загрузки.

Эти три машины имели много общего. Все они придерживались архитектуры, отделяющей команды обработки от команд работы с памятью, и делали упор на эффективную конвейерную обработку. Система команд разрабатывалась таким образом, чтобы выполнение любой команды занимало небольшое количество машинных тактов (предпочтительно один машинный такт). Сама логика выполнения команд с целью повышения производительности ориентировалась на аппаратную, а не на микропрограммную реализацию. Чтобы упростить логику декодирования команд использовались команды фиксированной длины и фиксированного формата.

Среди других особенностей RISC-архитектур следует отметить наличие достаточно большого регистрового файла (в типовых RISC-процессорах реализуются 32 или большее число регистров по сравнению с 8 - 16 регистрами в CISC-архитектурах), что позволяет большему объему данных храниться в регистрах на процессорном кристалле большее время и упрощает работу компилятора по распределению регистров под переменные. Для обработки, как правило, используются трехадресные команды, что помимо упрощения дешифрации дает возможность сохранять большее число переменных в регистрах без их последующей перезагрузки.

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

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

Следует отметить, что в последних разработках компании Intel (имеется в виду Pentium Pro и процессор следующего поколения Pentium II), а также ее последователей-конкурентов (AMD R5, Cyrix M1, NexGen Nx586 и др.) широко используются идеи, реализованные в RISC-микропроцессорах, так что многие различия между CISC и RISC стираются. Однако сложность архитектуры и системы команд x86 остается и является главным фактором, ограничивающим производительность процессоров на ее основе.

3.2. **4.2. Методы адресации и типы данных**

3.2.1. **4.2.1. Методы адресации**

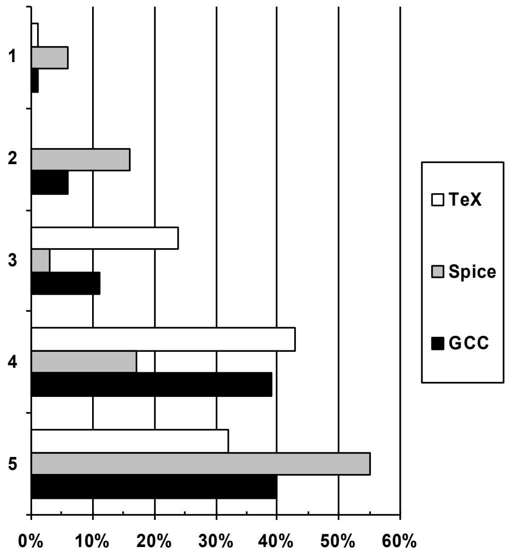
В машинах с регистрами общего назначения метод (или режим) адресации объектов, с которыми манипулирует команда, может задавать константу, регистр или ячейку памяти. Для обращения к ячейке памяти процессор, прежде всего, должен вычислить действительный или эффективный адрес памяти, который определяется заданным в команде методом адресации.

На рис. 4.1 представлены все основные методы адресации операндов, которые реализованы в компьютерах, рассмотренных в настоящем обзоре. Адресация

непосредственных данных и литеральных констант обычно рассматривается как один из методов адресации памяти (хотя значения данных, к которым в этом случае производятся обращения, являются частью самой команды и обрабатываются в общем потоке команд). Адресация регистров, как правило, рассматривается отдельно. В данном разделе методы адресации, связанные со счетчиком команд (адресация относительно счетчика команд) рассматриваются отдельно. Этот вид адресации используется главным образом для определения программных адресов в командах передачи управления.
На рисунке, на примере команды сложения (Add) приведены наиболее употребительные названия методов адресации, хотя при описании архитектуры в документации разные производители используют разные названия для этих методов. На этом рисунке знак "¬ " используется для обозначения оператора присваивания, а буква М обозначает память (Memory). Таким образом, M[R1] обозначает содержимое ячейки памяти, адрес которой определяется содержимым регистра R1.
Использование сложных методов адресации позволяет существенно сократить количество команд в программе, но при этом значительно увеличивается сложность аппаратуры. Возникает вопрос, а как часто эти методы адресации используются в реальных программах? На рис. 4.2 представлены результаты измерений частоты использования различных методов адресации на примере трех популярных программ (компилятора с языка Си GCC, текстового редактора TeX и САПР Spice), выполненных на компьютере VAX.

Метод адресации	**Пример команды**	**Смысл команды метода**	**Использование**
Регистровая	Add R4,R3	R4¬ R4+R5	Требуемое значение в регистре
Непосредственная или литеральная	Add R4,#3	R4¬ R4+3	Для задания констант
Базовая со смещением	Add R4,100(R1)	R4¬ R4+M[100+R1]	Для обращения к локальным переменным
Косвенная регистровая	Add R4,(R1)	R4¬ R4+M[R1]	Для обращения по указателю или вычисленному адресу
Индексная	Add R3,(R1+R2)	R3¬ R3+M[R1+R2]	Иногда полезна при работе с массивами: R1 - база, R3 - индекс
Прямая или абсолютная	Add R1,(1000)	R1¬ R1+M[1000]	Иногда полезна для обращения к статическим данным
Косвенная	Add R1,@(R3)	R1¬ R1+M[M[R3]]	Если R3-адрес указателя p, то выбирается значение по этому указателю
Автоинкрементная	Add R1,(R2)+	R1¬ R1+M[R2] R2¬ R2+d	Полезна для прохода в цикле по массиву с шагом: R2 - начало массива В каждом цикле R2 получает приращение d
Автодекрементная	Add R1,(R2)-	R2¬ R2-d R1¬ R1+M[R2]	Аналогична предыдущей Обе могут использоваться для реализации стека
Базовая индексная со смещением и масштабированием	Add R1,100(R2)[R3]	R1¬ R1+M[100]+R2+R3*d	Для индексации массивов

Рис. 4.1. Методы адресации



1. Косвенная (1%, 6%, 1%)
2. Базовая индексная со смещением и масштабированием (0%, 16%, 6%)
3. Регистровая (24%, 3%, 11%)
4. Непосредственная (43%, 17%, 39%)
5. Базовая со смещением (32%, 55%, 40%)

Рис. 4.2. Частота использования различных методов адресации на программах TeX, Spice, GCC

Из этого рисунка видно, что непосредственная адресация и базовая со смещением доминируют.

При этом основной вопрос, который возникает для метода базовой адресации со смещением, связан с длиной (разрядностью) смещения. Выбор длины смещения, в конечном счете, определяет длину команды. Результаты измерений показали, что в подавляющем большинстве случаев длина смещения не превышает16 разрядов.

Этот же вопрос важен и для непосредственной адресации. Непосредственная адресация используется при выполнении арифметических операций, операций сравнения, а также для загрузки констант в регистры. Результаты анализа статистики показывают, что в подавляющем числе случаев 16 разрядов оказывается вполне достаточно (хотя для вычисления адресов намного реже используются и более длинные константы).

Важным вопросом построения любой системы команд является оптимальное кодирование команд. Оно определяется количеством регистров и применяемых методов адресации, а также сложностью аппаратуры, необходимой для декодирования. Именно поэтому в современных RISC-архитектурах используются достаточно простые методы адресации, позволяющие резко упростить декодирование команд. Более сложные и редко встречающиеся в реальных программах методы адресации реализуются с помощью дополнительных команд, что, вообще говоря, приводит к увеличению размера программного кода. Однако такое увеличение длины программы с лихвой окупается возможностью простого увеличения тактовой частоты RISC-процессоров.

3.2.2. **4.2.2. Типы команд**

Команды традиционного машинного уровня можно разделить на несколько типов, которые показаны на рис. 4.3.

Тип операции	Примеры
Арифметические и логические	Целочисленные арифметические и логические операции: сложение, вычитание, логическое сложение, логическое

	умножение и т.д.
Пересылки данных	Операции загрузки/записи
Управление потоком команд	Безусловные и условные переходы, вызовы процедур и возвраты
Системные операции	Системные вызовы, команды управления виртуальной памятью и т.д.
Операции с плавающей точкой	Операции сложения, вычитания, умножения и деления над вещественными числами
Десятичные операции	Десятичное сложение, умножение, преобразование форматов и т.д.
Операции над строками	Пересылки, сравнения и поиск строк

Рис. 4.3. Основные типы команд

3.2.3. **4.2.3. Команды управления потоком команд**

В английском языке для указания команд безусловного перехода, как правило, используется термин *jump*, а для команд условного перехода - термин *branch*, хотя разные поставщики необязательно придерживаются этой терминологии. Например, компания Intel использует термин *jump* и для условных, и для безусловных переходов. Можно выделить четыре основных типа команд для управления потоком команд: условные переходы, безусловные переходы, вызовы процедур и возвраты из процедур.

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

При этом примерно 90% команд безусловного перехода выполняются относительно счетчика команд. Для команд перехода адрес перехода должен быть всегда заранее известным. Это не относится к адресам возврата, которые не известны во время компиляции программы и должны определяться во время ее работы. Наиболее простой способ определения адреса перехода заключается в указании его положения относительно текущего значения счетчика команд (с помощью смещения в команде), и такие переходы называются переходами относительно счетчика команд. Преимуществом такого метода адресации является то, что адреса переходов, как правило, расположены недалеко от текущего адреса выполняемой команды и указание относительно текущего значения счетчика команд требует небольшого количества бит в смещении. Кроме того, использование адресации относительно счетчика команд позволяет программе выполняться в любом месте памяти, независимо от того, куда она была загружена. То есть этот метод адресации позволяет автоматически создавать перемещаемые программы.

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

Одним из ключевых вопросов реализации команд перехода состоит в том, насколько далеко целевой адрес перехода находится от самой команды перехода? И на этот вопрос статистика использования команд дает ответ: в подавляющем большинстве случаев переход идет в пределах 3 - 7 команд относительно команды перехода, причем в 75% случаев выполняются переходы в направлении увеличения адреса, т.е. вперед по программе.

Поскольку большинство команд управления потоком команд составляют команды условного перехода, важным вопросом реализации архитектуры является определение условий перехода. Для этого используются три различных подхода. При первом из них в архитектуре процессора предусматривается специальный регистр, разряды которого соответствуют определенным кодам условий. Команды условного перехода проверяют эти условия в процессе своего выполнения. Преимуществом такого подхода является то, что иногда установка кода условия и переход по нему могут быть выполнены без дополнительных потерь времени, что, впрочем, бывает достаточно редко. А недостатками такого подхода является то, что, во-первых, появляются новые состояния машины, за которыми необходимо следить (упрятывать при прерывании и восстанавливать при возврате из него). Во-вторых, и что очень важно для современных высокоскоростных конвейерных архитектур, коды условий ограничивают порядок выполнения команд в потоке, поскольку их основное назначение заключается в передаче кода условия команде условного перехода.

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

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

Одним из наиболее заметных свойств большинства программ является преобладание в них сравнений на условие равно/неравно и сравнений с нулем. Поэтому в ряде архитектур такие команды выделяются в отдельный поднабор, особенно при использовании команд типа "сравнить и перейти".

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

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

3.2.4. 4.2.4. Типы и размеры операндов

Имеется два альтернативных метода определения типа операнда. В первом из них тип операнда может задаваться кодом операции в команде. Это наиболее употребительный способ задания типа операнда. Второй метод предполагает указание типа операнда с помощью тега, который хранится вместе с данными и интерпретируется аппаратурой во время выполнения операций над данными. Этот метод использовался, например, в машинах фирмы Burroughs, но в настоящее время он практически не применяется, и все современные процессоры пользуются первым методом.

Обычно тип операнда (например, целый, вещественный с одинарной точностью или символ) определяет и его размер. Однако часто процессоры работают с целыми числами длиною 8, 16, 32 или 64 бит. Как правило, целые числа представляются в дополнительном коде. Для задания символов (1 байт = 8 бит) в машинах компании IBM используется код EBCDIC, но в машинах других производителей почти повсеместно применяется кодировка ASCII. Еще до сравнительно недавнего времени каждый производитель процессоров пользовался своим собственным представлением вещественных чисел (чисел с плавающей точкой). Однако за последние несколько лет ситуация изменилась. Большинство поставщиков процессоров в настоящее время для представления вещественных чисел с одинарной и двойной точностью придерживаются стандарта IEEE 754.

В некоторых процессорах используются двоично-кодированные десятичные числа, которые представляются в упакованном и неупакованном форматах. Упакованный формат предполагает, что для кодирования цифр 0-9 используются 4 разряда и что две десятичные цифры упаковываются в каждый байт. В неупакованном формате байт содержит одну десятичную цифру, которая обычно изображается в символьном коде ASCII.

В большинстве процессоров, кроме того, реализуются операции над цепочками (строками) бит, байт, слов и двойных слов.

4. 5. Конвейерная обработка

4.1. 5.1. Что такое конвейерная обработка

Разработчики архитектуры компьютеров издавна прибегали к методам проектирования, известным под общим названием "совмещение операций", при котором аппаратура компьютера в любой момент времени выполняет одновременно более одной базовой операции. Этот общий метод включает два понятия: параллелизм и конвейеризацию. Хотя у них много общего и их зачастую трудно различать на практике, эти термины отражают два совершенно различных подхода. При параллелизме совмещение операций достигается путем воспроизведения в нескольких копиях аппаратной структуры. Высокая производительность достигается за счет одновременной работы всех элементов структур, осуществляющих решение различных частей задачи.

Конвейеризация (или конвейерная обработка) в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры. Так обработку любой машинной команды можно разделить на несколько этапов (несколько ступеней), организовав передачу данных от одного этапа к следующему. При этом конвейерную обработку можно использовать для совмещения этапов выполнения разных команд. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд. Конвейерная обработка такого рода широко применяется во всех современных быстродействующих процессорах.

4.2. 5.2. Простейшая организация конвейера и оценка его производительности

Для иллюстрации основных принципов построения процессоров мы будем использовать простейшую архитектуру, содержащую 32 целочисленных регистра общего назначения (R0,...,R31), 32 регистра плавающей точки (F0,...,F31) и счетчик команд PC. Будем считать, что набор команд нашего процессора включает типичные арифметические и логические операции, операции с плавающей точкой, операции пересылки данных, операции управления потоком команд и системные операции. В арифметических командах используется трехадресный формат, типичный для RISC-процессоров, а для обращения к памяти используются операции загрузки и записи содержимого регистров в память.

Выполнение типичной команды можно разделить на следующие этапы:

- • выборка команды - IF (по адресу, заданному счетчиком команд, из памяти извлекается команда);
- • декодирование команды / выборка операндов из регистров - ID;
- • выполнение операции / вычисление эффективного адреса памяти - EX;
- • обращение к памяти - MEM;
- • запоминание результата - WB.

На рис. 5.1 представлена схема простейшего процессора, выполняющего указанные выше этапы выполнения команд без совмещения. Чтобы конвейеризовать эту схему, мы можем просто разбить выполнение команд на указанные выше этапы, отведя для выполнения каждого этапа один такт синхронизации, и начинать в каждом такте выполнение новой команды. Естественно, для хранения промежуточных результатов каждого этапа необходимо использовать регистровые станции. На рис. 5.2 показана схема процессора с промежуточными регистровыми станциями, которые обеспечивают передачу данных и управляющих сигналов с одной ступени конвейера на следующую. Хотя общее время выполнения одной команды в таком конвейере будет составлять пять тактов, в каждом такте аппаратура будет выполнять в совмещенном режиме пять различных команд.

Работу конвейера можно условно представить в виде сдвинутых во времени схем процессора (рис. 5.3). Этот рисунок хорошо отражает совмещение во времени выполнения различных этапов команд. Однако чаще для представления работы конвейера используются временные диаграммы (рис. 5.4), на которых обычно изображаются выполняемые команды, номера тактов и этапы выполнения команд.

Конвейеризация увеличивает пропускную способность процессора (количество команд, завершающихся в единицу времени), но она не сокращает время выполнения отдельной команды. В действительности, она даже несколько увеличивает время выполнения каждой команды из-за накладных расходов, связанных с управлением регистровыми станциями. Однако увеличение пропускной способности означает, что программа будет выполняться быстрее по сравнению с простой неконвейерной схемой.

Тот факт, что время выполнения каждой команды в конвейере не уменьшается, накладывает некоторые ограничения на практическую длину конвейера. Кроме ограничений, связанных с задержкой конвейера, имеются также ограничения, возникающие в результате несбалансированности задержки на каждой его ступени и из-за накладных расходов на конвейеризацию. Частота синхронизации не может быть выше, а, следовательно, такт синхронизации не может быть меньше, чем время, необходимое для работы наиболее медленной ступени конвейера. Накладные расходы на организацию конвейера возникают из-за задержки сигналов в конвейерных регистрах (защелках) и из-за перекосов сигналов синхронизации. Конвейерные регистры к длительности такта добавляют время установки и задержку распространения сигналов. В предельном случае длительность такта можно уменьшить до суммы накладных расходов и перекоса сигналов синхронизации, однако при этом в такте не останется времени для выполнения полезной работы по преобразованию информации.

В качестве примера рассмотрим неконвейерную машину с пятью этапами выполнения операций, которые имеют длительность 50, 50, 60, 50 и 50 нс соответственно (рис. 5.5). Пусть накладные расходы на организацию конвейерной обработки составляют 5 нс. Тогда среднее время выполнения команды в неконвейерной машине будет равно 260 нс. Если же используется конвейерная организация, длительность такта будет равна длительности самого медленного этапа обработки плюс накладные расходы, т.е. 65 нс. Это время соответствует среднему времени выполнения команды в конвейере. Таким образом, ускорение, полученное в результате конвейеризации, будет равно:

$$\frac{\textbf{Среднее время выполнения команды в неконвейерном режиме}}{\textbf{Среднее время выполнения команды в неконвейерном режиме}} = \frac{\mathbf{260}}{\mathbf{65}} = \mathbf{4}$$

Конвейеризация эффективна только тогда, когда загрузка конвейера близка к полной, а скорость подачи новых команд и операндов соответствует максимальной производительности конвейера. Если произойдет задержка, то параллельно будет выполняться меньше операций и суммарная производительность снизится. Такие задержки могут возникать в результате возникновения конфликтных ситуаций.

Рис. 5.1. Схема неконвейерного целочисленного устройства

Рис. 5.2. Схема конвейерного целочисленного устройства

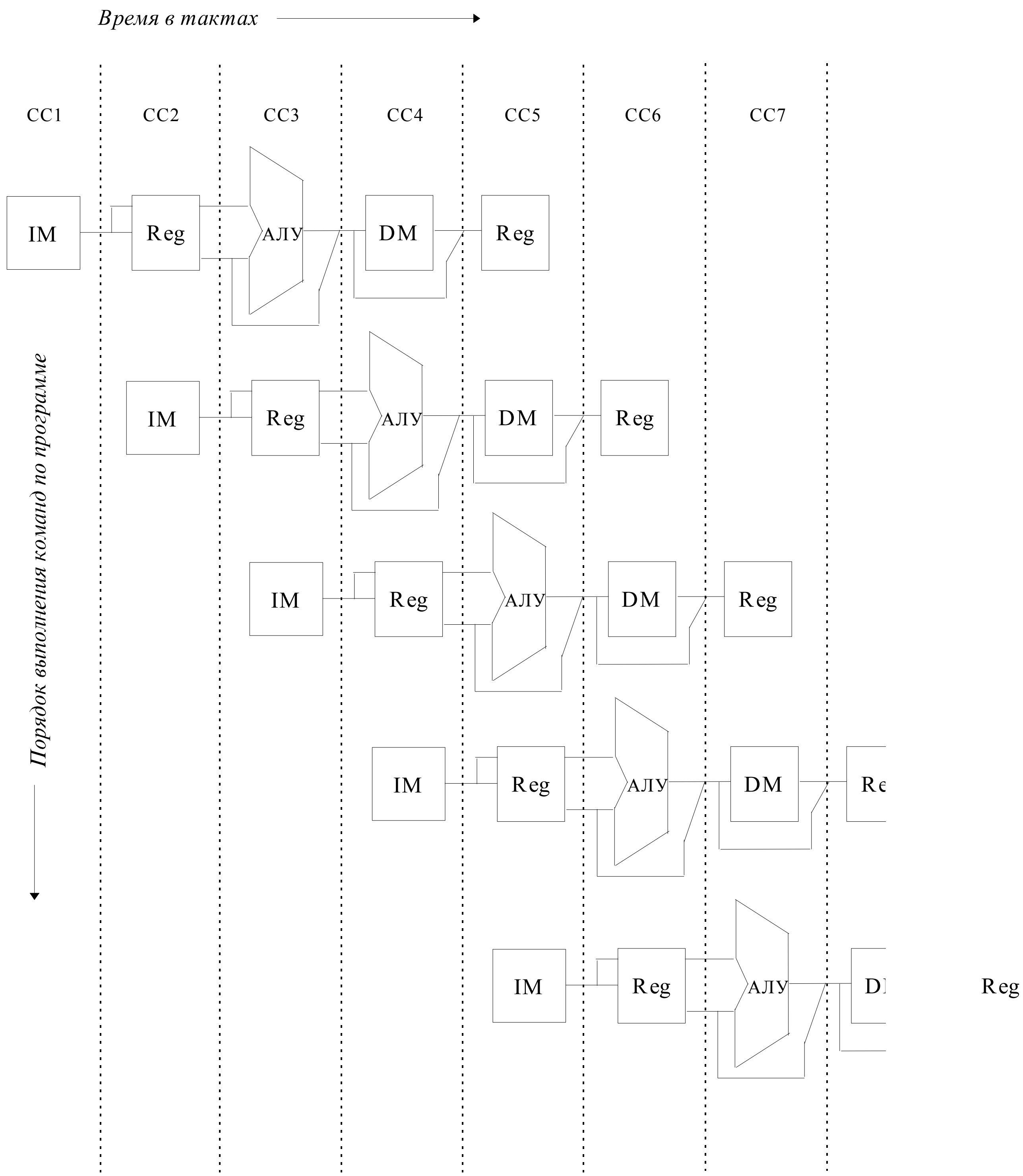
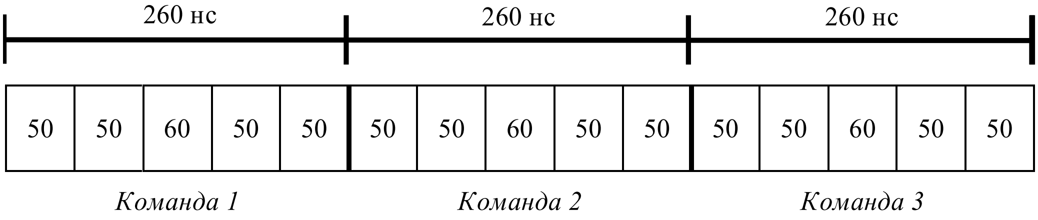


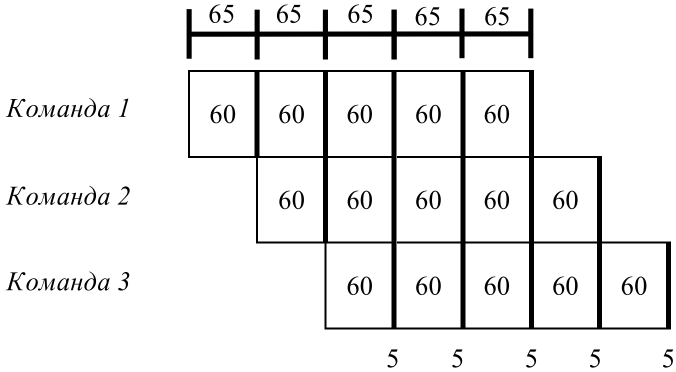
Рис. 5.3. Представление о работе конвейера

Номер команды	Номер такта								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Команда i	IF	ID	EX	MEM	WB				
Команда i+1		IF	ID	EX	MEM	WB			
Команда i+2			IF	ID	EX	MEM	WB		
Команда i+3				IF	ID	EX	MEM	WB	
Команда i+4					IF	ID	EX	MEM	WB

Рис. 5.4. Диаграмма работы простейшего конвейера



Неконвейерное выполнение



Конвейерное выполнение

Рис. 5.5. Эффект конвейеризации при выполнении 3-х команд - четырехкратное ускорение

Существуют три класса конфликтов:

1. 1. Структурные конфликты, которые возникают из-за конфликтов по ресурсам, когда аппаратные средства не могут поддерживать все возможные комбинации команд в режиме одновременного выполнения с совмещением.
2. 2. Конфликты по данным, возникающие в случае, когда выполнение одной команды зависит от результата выполнения предыдущей команды.
3. 3. Конфликты по управлению, которые возникают при конвейеризации команд переходов и других команд, которые изменяют значение счетчика команд.

Конфликты в конвейере приводят к необходимости приостановки выполнения команд (pipeline stall). Обычно в простейших конвейерах, если приостанавливается какая-либо команда, то все следующие за ней команды также приостанавливаются. Команды, предшествующие приостановленной, могут продолжать выполняться, но во время приостановки не выбирается ни одна новая команда.

4.3. **5.3. Структурные конфликты и способы их минимизации**

Совмещенный режим выполнения команд в общем случае требует конвейеризации функциональных устройств и дублирования ресурсов для разрешения всех возможных комбинаций команд в конвейере. Если какая-нибудь комбинация команд не может быть принята из-за конфликта по ресурсам, то говорят, что в машине имеется структурный конфликт. Наиболее типичным примером машин, в которых возможно появление структурных конфликтов, являются машины с не полностью конвейерными функциональными устройствами. Время работы такого устройства может составлять несколько тактов синхронизации конвейера. В этом случае последовательные команды, которые используют данное функциональное устройство, не могут поступать в него в каждом такте. Другая возможность появления структурных конфликтов связана с недостаточным дублированием некоторых ресурсов, что препятствует выполнению произвольной последовательности команд в конвейере без его приостановки. Например, машина может иметь только один порт записи в регистровый файл, но при определенных обстоятельствах конвейеру может потребоваться выполнить две записи в регистровый файл в одном такте. Это также приведет к структурному конфликту. Когда последовательность команд наталкивается на такой конфликт, конвейер приостанавливает выполнение одной из команд до тех пор, пока не станет доступным требуемое устройство.

Структурные конфликты возникают, например, и в машинах, в которых имеется единственный конвейер памяти для команд и данных (рис. 5.6). В этом случае, когда одна команда содержит обращение к памяти за данными, оно будет конфликтовать с выборкой более поздней команды из памяти. Чтобы разрешить эту ситуацию, можно просто приостановить конвейер на один такт, когда происходит обращение к памяти за данными. Подобная приостановка часто называются "конвейерным пузырем" (pipeline bubble) или просто пузырем, поскольку пузырь проходит по конвейеру, занимая место, но не выполняя никакой полезной работы.

При всех прочих обстоятельствах, машина без структурных конфликтов будет всегда иметь более низкий CPI (среднее число тактов на выдачу команды). Возникает вопрос: почему разработчики допускают наличие структурных конфликтов? Для этого имеются две причины: снижение стоимости и уменьшение задержки устройства. Конвейеризация всех функциональных устройств может оказаться слишком дорогой. Машины, допускающие два обращения к памяти в одном такте, должны иметь удвоенную пропускную способность памяти, например, путем организации раздельных кэшей для команд и данных. Аналогично, полностью конвейерное устройство деления с плавающей точкой требует огромного количества вентилей. Если структурные конфликты не будут возникать слишком часто, то может быть и не стоит платить за то, чтобы их обойти. Как правило, можно разработать неконвейерное, или не полностью конвейерное устройство, имеющее меньшую общую задержку, чем полностью конвейерное. Например, разработчики устройств с плавающей точкой компьютеров CDC7600 и MIPS R2010 предпочли иметь меньшую задержку выполнения операций вместо полной их конвейеризации.

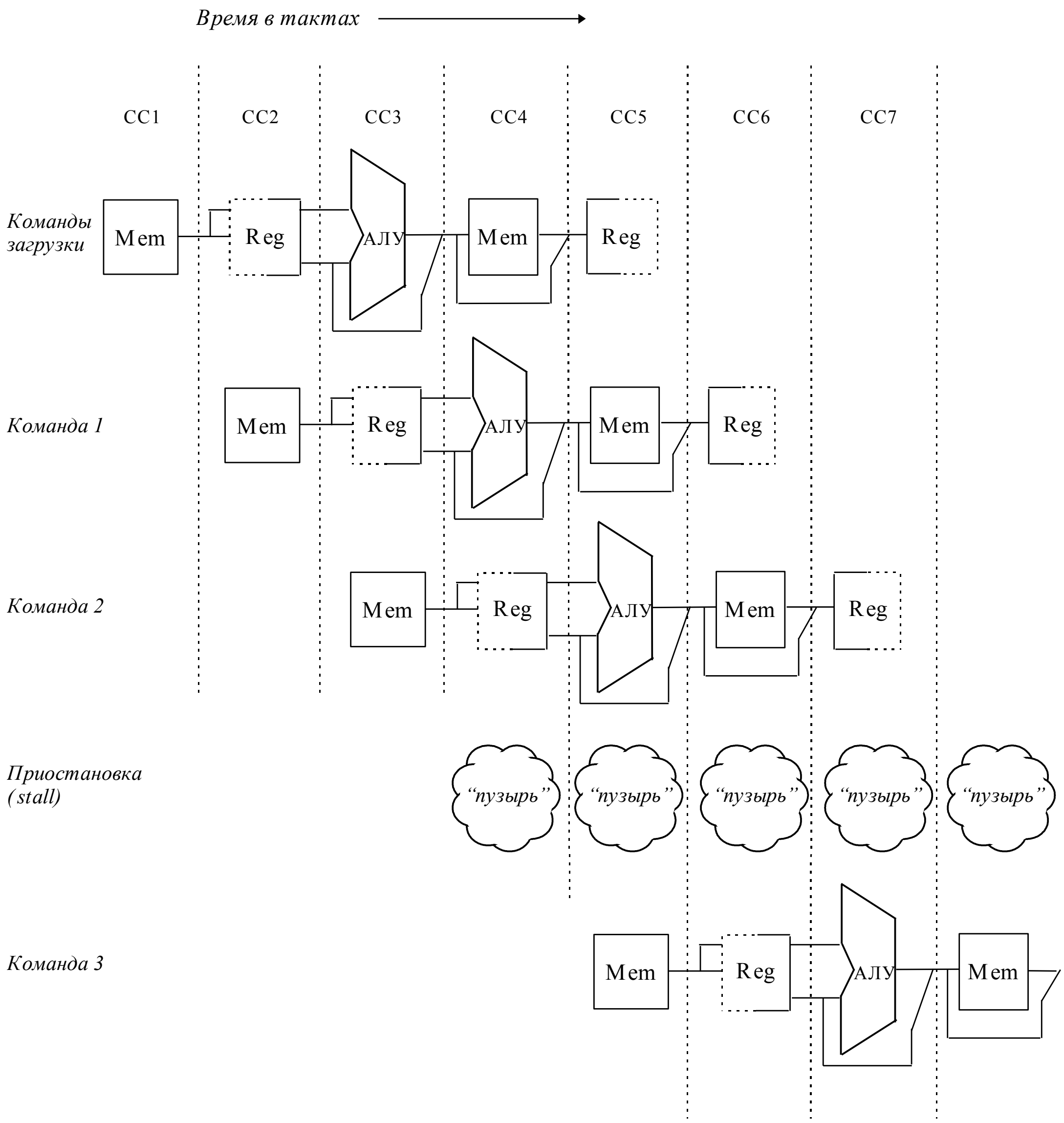


Рис. 5.6, а. Пример структурного конфликта при реализации памяти с одним портом

Команда	**Номер такта**									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Команда загрузки	IF	ID	EX	MEM	WB					
Команда 1		IF	ID	EX	MEM	WB				

Команда 2	IF	ID	EX	MEM	WB			
Команда 3		stall	IF	ID	EX	MEM	WB	
Команда 4				IF	ID	EX	MEM	WB
Команда 5					IF	ID	EX	MEM
Команда 6						IF	ID	EX

Рис. 5.6, б. Диаграмма работы конвейера при структурном конфликте

4.4. **5.4. Конфликты по данным, остановы конвейера и реализация механизма обходов**

Одним из факторов, который оказывает существенное влияние на производительность конвейерных систем, являются межкомандные логические зависимости. Такие зависимости в большой степени ограничивают потенциальный параллелизм смежных операций, обеспечиваемый соответствующими аппаратными средствами обработки. Степень влияния этих зависимостей определяется как архитектурой процессора (в основном, структурой управления конвейером команд и параметрами функциональных устройств), так и характеристиками программ.

Конфликты по данным возникают в том случае, когда применение конвейерной обработки может изменить порядок обращений за операндами так, что этот порядок будет отличаться от порядка, который наблюдается при последовательном выполнении команд на неконвейерной машине. Рассмотрим конвейерное выполнение последовательности команд на рисунке 5.7.

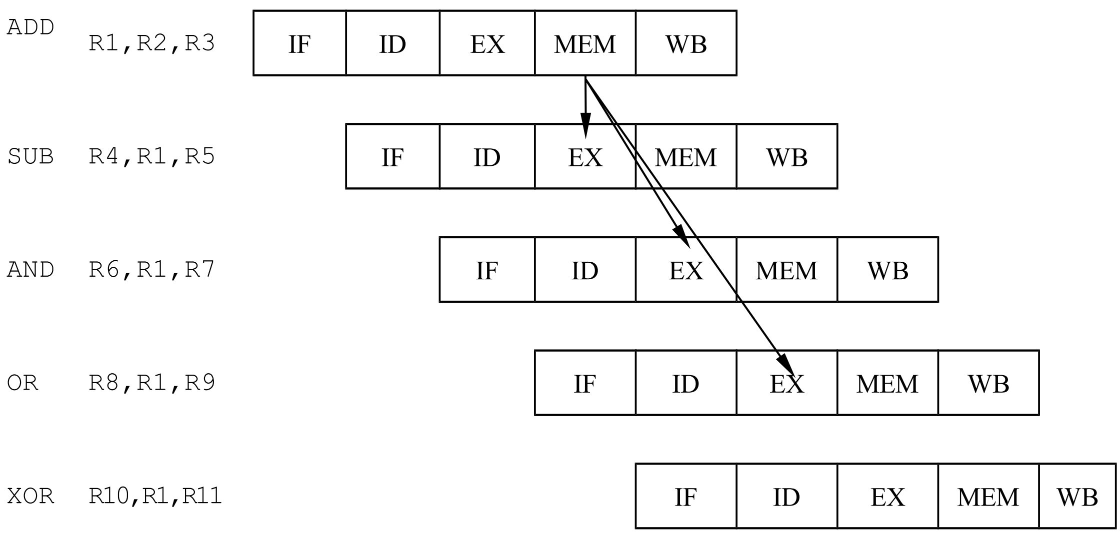
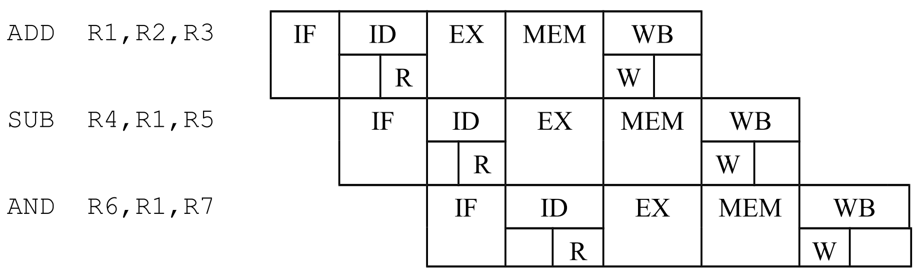


Рис. 5.7, а. Последовательность команд в конвейере и ускоренная пересылка данных (data forwarding, data bypassing, short circuiting)



OR R8,R1,R9	IF	ID	EX	MEM	WB	
		R			W	
XOR R10,R1,R11		IF	ID	EX	MEM	WB
			R			W

Рис. 5.7, б. Совмещение чтения и записи регистров в одном такте

В этом примере все команды, следующие за командой ADD, используют результат ее выполнения. Команда ADD записывает результат в регистр R1, а команда SUB читает это значение. Если не предпринять никаких мер для того, чтобы предотвратить этот конфликт, команда SUB прочитает неправильное значение и попытается его использовать. На самом деле значение, используемое командой SUB, является даже неопределенным: хотя логично предположить, что SUB всегда будет использовать значение R1, которое было присвоено какой-либо командой, предшествовавшей ADD, это не всегда так. Если произойдет прерывание между командами ADD и SUB, то команда ADD завершится, и значение R1 в этой точке будет соответствовать результату ADD. Такое непрогнозируемое поведение очевидно неприемлемо.
Проблема, поставленная в этом примере, может быть разрешена с помощью достаточно простой аппаратной техники, которая называется пересылкой или продвижением данных (data forwarding), обходом (data bypassing), иногда закороткой (short-circuiting). Эта аппаратура работает следующим образом. Результат операции АЛУ с его выходного регистра всегда снова подается назад на входы АЛУ. Если аппаратура обнаруживает, что предыдущая операция АЛУ записывает результат в регистр, соответствующий источнику операнда для следующей операции АЛУ, то логические схемы управления выбирают в качестве входа для АЛУ результат, поступающий по цепи "обхода", а не значение, прочитанное из регистрового файла (рис. 5.8).

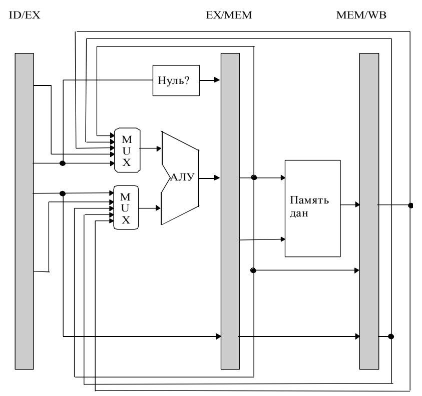


Рис. 5.8. АЛУ с цепями обхода и ускоренной пересылки

Эта техника "обходов" может быть обобщена для того, чтобы включить передачу результата прямо в то функциональное устройство, которое в нем нуждается: результат с выхода одного устройства "пересылается" на вход другого, а не с выхода некоторого устройства только на его вход.

Классификация конфликтов по данным

Конфликт возникает везде, где имеет место зависимость между командами, и они расположены по отношению друг к другу достаточно близко так, что совмещение операций, происходящее при конвейеризации, может привести к изменению порядка обращения к операндам. В нашем примере был проиллюстрирован конфликт, происходящий с регистровыми операндами, но для пары команд возможно появление зависимостей при записи или чтении одной и той же ячейки памяти. Однако, если все обращения к памяти выполняются в строгом порядке, то появление такого типа конфликтов предотвращается.

Известны три возможных конфликта по данным в зависимости от порядка операций чтения и записи. Рассмотрим две команды i и j, при этом i предшествует j. Возможны следующие конфликты:

- • RAW (чтение после записи) - j пытается прочитать операнд-источник данных прежде, чем i туда запишет. Таким образом, j может некорректно получить старое значение. Это наиболее общий тип конфликтов, способ их преодоления с помощью механизма "обходов" рассмотрен ранее.
- • WAR (запись после чтения) - j пытается записать результат в приемник прежде, чем он считывается оттуда командой i, так что i может некорректно получить новое значение. Этот тип конфликтов как правило не возникает в системах с централизованным управлением потоком команд, обеспечивающих выполнение команд в порядке их поступления, так как последующая запись всегда выполняется позже, чем предшествующее считывание. Особенно часто конфликты такого рода могут возникать в системах, допускающих выполнение команд не в порядке их расположения в программном коде.
- • WAW (запись после записи) - j пытается записать операнд прежде, чем будет записан результат команды i, т.е. записи заканчиваются в неверном порядке, оставляя в приемнике значение, записанное командой i, а не j. Этот тип конфликтов присутствует только в конвейерах, которые выполняют запись со многих ступеней (или позволяют команде выполняться даже в случае, когда предыдущая приостановлена).

Конфликты по данным, приводящие к приостановке конвейера

К сожалению не все потенциальные конфликты по данным могут обрабатываться с помощью механизма "обходов". Рассмотрим следующую последовательность команд (рис. 5.9):

Команда	IF	ID	EX	MEM	WB					
`LW R1,32(R6)`		IF	ID	EX	MEM	WB				
`ADD R4,R1,R7`			IF	ID	stall	EX	MEM	WB		
`SUB R5,R1,R8`				IF	stall	ID	EX	MEM	WB	
`AND R6,R1,R7`					stall	IF	ID	EX	MEM	WB

Рис. 5.9. Последовательность команд с приостановкой конвейера

Этот случай отличается от последовательности подряд идущих команд АЛУ. Команда загрузки (LW) регистра R1 из памяти имеет задержку, которая не может быть устранена обычной "пересылкой". Вместо этого нам нужна дополнительная аппаратура, называемая аппаратурой внутренних блокировок конвейера (pipeline interlook), чтобы обеспечить корректное выполнение примера. Вообще такого рода аппаратура обнаруживает конфликты и приостанавливает конвейер до тех пор, пока существует конфликт. В этом случае эта аппаратура приостанавливает конвейер начиная с команды, которая хочет использовать данные в то время, когда предыдущая команда, результат которой является операндом для нашей, вырабатывает этот результат. Эта аппаратура вызывает приостановку конвейера или появление "пузыря" точно также, как и в случае структурных конфликтов.

Методика планирования компилятора для устранения конфликтов по данным

Многие типы приостановок конвейера могут происходить достаточно часто. Например, для оператора A = B + C компилятор скорее всего сгенерирует следующую последовательность команд (рис.5.10):

LW R1,B	IF	ID	EX	MEM	WB				
LW R2,C		IF	ID	EX	MEM	WB			
ADD R3,R1,R2			IF	ID	stall	EX	MEM	WB	
SW A,R3				IF	stall	ID	EX	MEM	WB

Рис. 5.10. Конвейерное выполнение оператора A = B + C

Очевидно, выполнение команды ADD должно быть приостановлено до тех пор, пока не станет доступным поступающий из памяти операнд C. Дополнительной задержки выполнения команды SW не произойдет в случае применения цепей обхода для пересылки результата операции АЛУ непосредственно в регистр данных памяти для последующей записи.

Для данного простого примера компилятор никак не может улучшить ситуацию, однако в ряде более общих случаев он может реорганизовать последовательность команд так, чтобы избежать приостановок конвейера. Эта техника, называемая планированием загрузки конвейера (pipeline scheduling) или планированием потока команд (instruction scheduling), использовалась начиная с 60-х годов и стала особой областью интереса в 80-х годах, когда конвейерные машины стали более распространенными.

Пусть, например, имеется последовательность операторов: a = b + c; d = e - f;

Как сгенерировать код, не вызывающий остановок конвейера? Предполагается, что задержка загрузки из памяти составляет один такт. Ответ очевиден (рис. 5.11):

Неоптимизированная последовательность команд	*Оптимизированная последовательность команд*
LW R_b,b	LW R_b,b
LW R_c,c	LW R_c,c
ADD R_a,R_b,R_c	LW R_e,e
SW a,R_a	ADD R_a,R_b,R_c
LW R_e,e	LW R_f,f
LW R_f,f	SW a,R_a
SUB R_d,R_e,R_f	SUB R_d,R_e,R_f
SW d,R_d	SW d,R_d

Рис. 5.11. Пример устранения конфликтов компилятором

В результате устранены обе блокировки (командой LW R_c,c команды ADD R_a,R_b,R_c и командой LW R_f,f команды SUB R_d,R_e,R_f). Имеется зависимость между операцией АЛУ и операцией записи в память, но структура конвейера допускает пересылку результата с помощью цепей "обхода". Заметим, что использование разных регистров для первого и второго операторов было достаточно важным для реализации такого правильного планирования. В частности, если переменная e была бы загружена в тот же самый регистр, что b или c, такое планирование не было бы корректным. В общем случае планирование конвейера может требовать увеличенного количества регистров. Такое увеличение может оказаться особенно существенным для машин, которые могут выдавать на выполнение несколько команд в одном такте.

Многие современные компиляторы используют технику планирования команд для улучшения производительности конвейера. В простейшем алгоритме компилятор просто планирует распределение команд в одном и том же базовом блоке. Базовый блок представляет собой линейный участок последовательности программного кода, в котором отсутствуют команды перехода, за исключением начала и конца участка (переходы внутрь этого участка тоже должны отсутствовать). Планирование такой последовательности команд осуществляется достаточно просто, поскольку компилятор знает, что каждая команда в блоке будет выполняться, если выполняется первая из них, и можно просто построить граф зависимостей этих команд и упорядочить их так, чтобы минимизировать приостановки конвейера. Для простых конвейеров стратегия планирования на основе базовых блоков вполне удовлетворительна. Однако когда конвейеризация становится более интенсивной и действительные задержки конвейера растут, требуются более сложные алгоритмы планирования.

К счастью, существуют аппаратные методы, позволяющие изменить порядок выполнения команд программы так, чтобы минимизировать приостановки конвейера. Эти методы получили общее название методов динамической оптимизации (в англоязычной литературе в последнее время часто применяются также термины "out-of-order execution" - неупорядоченное выполнение и "out-of-order issue" - неупорядоченная выдача). Основными средствами динамической оптимизации являются:

1. 1. Размещение схемы обнаружения конфликтов в возможно более низкой точке конвейера команд так, чтобы позволить команде продвигаться по конвейеру до тех пор, пока ей реально не потребуется операнд, являющийся также результатом логически более ранней, но еще не завершившейся команды. Альтернативным подходом является централизованное обнаружение конфликтов на одной из ранних ступеней конвейера.
2. 2. Буферизация команд, ожидающих разрешения конфликта, и выдача последующих, логически не связанных команд, в "обход" буфера. В этом случае команды могут выдаваться на выполнение не в том порядке, в котором они расположены в программе, однако аппаратура обнаружения и устранения конфликтов между логически связанными командами обеспечивает получение результатов в соответствии с заданной программой.
3. 3. Соответствующая организация коммутирующих магистралей, обеспечивающая засылку результата операции непосредственно в буфер, хранящий логически зависимую команду, задержанную из-за конфликта, или непосредственно на вход функционального устройства до того, как этот результат будет записан в регистровый файл или в память (short-circuiting, data forwarding, data bypassing - методы, которые были рассмотрены ранее).

Еще одним аппаратным методом минимизации конфликтов по данным является метод переименования регистров (register renaming). Он получил свое название от широко применяющегося в компиляторах метода переименования - метода размещения данных, способствующего сокращению числа зависимостей и тем самым увеличению производительности при отображении необходимых исходной программе объектов (например, переменных) на аппаратные ресурсы (например, ячейки памяти и регистры).

При аппаратной реализации метода переименования регистров выделяются логические регистры, обращение к которым выполняется с помощью соответствующих полей команды, и физические регистры, которые размещаются в аппаратном регистровом файле процессора. Номера логических регистров динамически отображаются на номера физических регистров посредством таблиц отображения, которые обновляются

после декодирования каждой команды. Каждый новый результат записывается в новый физический регистр. Однако предыдущее значение каждого логического регистра сохраняется и может быть восстановлено в случае, если выполнение команды должно быть прервано из-за возникновения исключительной ситуации или неправильного предсказания направления условного перехода.
В процессе выполнения программы генерируется множество временных регистровых результатов. Эти временные значения записываются в регистровые файлы вместе с постоянными значениями. Временное значение становится новым постоянным значением, когда завершается выполнение команды (фиксируется ее результат). В свою очередь, завершение выполнения команды происходит, когда все предыдущие команды успешно завершились в заданном программой порядке. Метод переименования регистров упрощает контроль зависимостей по данным.

4.5. **5.5. Сокращение потерь на выполнение команд перехода и минимизация конфликтов по управлению**

Конфликты по управлению могут вызывать даже большие потери производительности конвейера, чем конфликты по данным. Когда выполняется команда условного перехода, она может либо изменить, либо не изменить значение счетчика команд. Если команда условного перехода заменяет счетчик команд значением адреса, вычисленного в команде, то переход называется выполняемым; в противном случае, он называется невыполняемым.
Простейший метод работы с условными переходами заключается в приостановке конвейера как только обнаружена команда условного перехода до тех пор, пока она не достигнет ступени конвейера, которая вычисляет новое значение счетчика команд (рис. 5.12). Такие приостановки конвейера из-за конфликтов по управлению должны реализовываться иначе, чем приостановки из-за конфликтов по данным, поскольку выборка команды, следующей за командой условного перехода, должна быть выполнена как можно быстрее, как только мы узнаем окончательное направление команды условного перехода.

Команды перехода	IF	ID	EX	MEM	WB					
Следующая команда		IF	stall	stall	IF	ID	EX	MEM	WB	
Следующая команда +1			stall	stall	stall	IF	ID	EX	MEM	WB
Следующая команда +2				stall	stall	stall	IF	ID	EX	MEM
Следующая команда +3					stall	stall	stall	IF	ID	EX
Следующая команда +4						stall	stall	stall	IF	ID
Следующая команда +5							stall	stall	stall	IF

Рис. 5.12. Приостановка конвейера при выполнении команды условного перехода

Например, если конвейер будет приостановлен на три такта на каждой команде условного перехода, то это может существенно отразиться на производительности машины. При частоте команд условного перехода в программах, равной 30% и идеальном CPI, равным 1, машина с приостановками условных переходов достигает примерно только половины ускорения, получаемого за счет конвейерной организации. Таким образом, снижение потерь от условных переходов становится критическим вопросом. Число тактов, теряемых при приостановках из-за условных переходов, может быть уменьшено двумя способами:

1. 1. Обнаружением является ли условный переход выполняемым или невыполняемым на более ранних ступенях конвейера.
2. 2. Более ранним вычислением значения счетчика команд для выполняемого перехода (т.е. вычислением целевого адреса перехода).

Реализация этих условий требует модернизации исходной схемы конвейера (рис. 5.13).

В некоторых машинах конфликты из-за условных переходов являются даже еще более дорогостоящими по количеству тактов, чем в нашем примере, поскольку время на оценку условия перехода и вычисление адреса перехода может быть даже большим. Например, машина с раздельными ступенями декодирования и выборки команд возможно будет иметь задержку условного перехода (длительность конфликта по управлению), которая по крайней мере на один такт длиннее. Многие компьютеры VAX имеют задержки условных переходов в четыре и более тактов, а большие машины с глубокими конвейерами имеют потери по условным переходам, равные шести или семи тактам. В общем случае, чем глубина конвейера больше, тем больше потери на командах условного перехода, исчисляемые в тактах. Конечно эффект снижения относительной производительности при этом зависит от общего CPI машины. Машины с высоким CPI могут иметь условные переходы большей длительности, поскольку процент производительности машины, которая будет потеряна из-за условных переходов, меньше.

Снижение потерь на выполнение команд условного перехода

Имеется несколько методов сокращения приостановок конвейера, возникающих из-за задержек выполнения условных переходов. В данном разделе обсуждаются четыре простые схемы, используемые во время компиляции. В этих схемах прогнозирование направления перехода выполняется статически, т.е. прогнозируемое направление перехода фиксируется для каждой команды условного перехода на все время выполнения программы. После обсуждения этих схем мы исследуем вопрос о правильности предсказания направления перехода компиляторами, поскольку все эти схемы основаны на такой технологии. В следующей главе мы рассмотрим более мощные схемы, используемые компиляторами (такие, например, как разворачивание циклов), которые уменьшают частоту команд условных переходов при реализации циклов, а также динамические, аппаратно реализованные схемы прогнозирования.

Метод выжидания

Простейшая схема обработки команд условного перехода заключается в замораживании или подавлении операций в конвейере, путем блокировки выполнения любой команды, следующей за командой условного перехода, до тех пор, пока не станет известным направление перехода. Рис. 5.12 отражал именно такой подход. Привлекательность такого решения заключается в его простоте.

Метод возврата

Более хорошая, и не на много более сложная схема состоит в том, чтобы прогнозировать условный переход как невыполняемый. При этом аппаратура должна просто продолжать выполнение программы, как если бы условный переход вовсе не выполнялся. В этом случае необходимо позаботиться о том, чтобы не изменить состояние машины до тех пор, пока направление перехода не станет окончательно известным. В некоторых машинах эта схема с невыполняемыми по прогнозу условными переходами реализована путем продолжения выборки команд, как если бы условный переход был обычной командой. Поведение конвейера выглядит так, как будто ничего необычного не происходит. Однако, если условный переход на самом деле выполняется, то необходимо просто очистить конвейер от команд, выбранных вслед за командой условного перехода и заново повторить выборку команд (рис. 5.14).

Невыполняемый условный переход	IF	ID	EX	MEM	WB				
Команда i+1		IF	ID	EX	MEM	WB			
Команда i+2			IF	ID	EX	MEM	WB		
Команда i+3				IF	ID	EX	MEM	WB	
Команда i+4					IF	ID	EX	MEM	WB

Выполняемый условный переход	IF	ID	EX	MEM	WB				
Команда i+1		IF	ID	EX	MEM	WB			
Команда i+2			stall	IF	ID	EX	MEM	WB	
Команда i+3				stall	IF	ID	EX	MEM	WB
Команда i+4					stall	IF	ID	EX	MEM

Рис. 5.14. Диаграмма работы модернизированного конвейера

Альтернативная схема прогнозирует переход как выполняемый. Как только команда условного перехода декодирована и вычислен целевой адрес перехода, мы предполагаем, что переход выполняемый, и осуществляем выборку команд и их выполнение, начиная с целевого адреса. Если мы не знаем целевой адрес перехода раньше, чем узнаем окончательное направление перехода, у этого подхода нет никаких преимуществ. Если бы условие перехода зависело от непосредственно предшествующей команды, то произошла бы приостановка конвейера из-за конфликта по данным для регистра, который является условием перехода, и мы бы узнали сначала целевой адрес. В таких случаях прогнозировать переход как выполняемый было бы выгодно. Дополнительно в некоторых машинах (особенно в машинах с устанавливаемыми по умолчанию кодами условий или более мощным (а потому и более медленным) набором условий перехода) целевой адрес перехода известен раньше окончательного направления перехода, и схема прогноза перехода как выполняемого имеет смысл.

Задержанные переходы

Четвертая схема, которая используется в некоторых машинах, называется "задержанным переходом". В задержанном переходе такт выполнения с задержкой перехода длиною n есть:

команда условного перехода
следующая команда 1
следующая команда 2
.....
следующая команда n
целевой адрес при выполняемом переходе

Команды 1 - n находятся в слотах (временных интервалах) задержанного перехода. Задача программного обеспечения заключается в том, чтобы сделать команды, следующие за командой перехода, действительными и полезными. Аппаратура гарантирует реальное выполнение этих команд перед выполнением собственно перехода. Здесь используются несколько приемов оптимизации.

На рис. 5.15, а показаны три случая, при которых может планироваться задержанный переход. В верхней части рисунка для каждого случая показана исходная последовательность команд, а в нижней части - последовательность команд, полученная в результате планирования. В случае (a) слот задержки заполняется независимой командой, находящейся перед командой условного перехода. Это наилучший выбор. Стратегии (b) и (c) используются, если применение стратегии (a) невозможно.

В последовательностях команд для случаев (b) и (c) использование содержимого регистра R1 в качестве условия перехода препятствует перемещению команды ADD (которая записывает результат в регистр R1) за команду перехода. В случае (b) слот задержки заполняется командой, находящейся по целевому адресу команды перехода. Обычно такую команду приходится копировать, поскольку к ней возможны обращения и из других частей программы. Стратегии (b) отдается предпочтение, когда с высокой вероятностью переход является выполняемым, например, если это переход на начало цикла.

Наконец, слот задержки может заполняться командой, находящейся между командой невыполняемого перехода и командой, находящейся по целевому адресу, как в случае (c). Чтобы подобная оптимизация была законной, необходимо, чтобы можно было все-таки выполнить команду SUB, если переход пойдет не по прогнозируемому направлению. При этом мы предполагаем, что команда SUB выполнит ненужную работу, но вся программа при этом будет выполняться корректно. Это, например, может быть в случае, если регистр R4 используется только для временного хранения промежуточных результатов вычислений, когда переход выполняется не по прогнозируемому направлению.

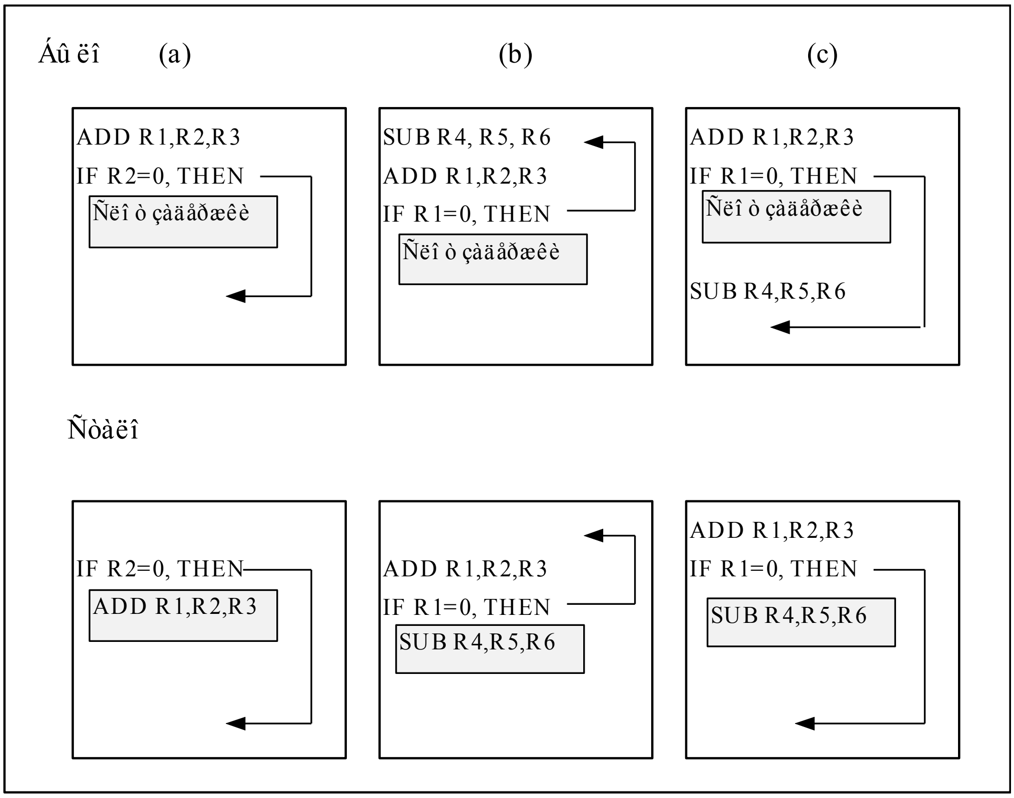


Рис. 5.15, а. Требования к переставляемым командам при планировании задержанного перехода

Рис. 5.15, б показывает различные ограничения для всех этих схем планирования условных переходов, а также ситуации, в которых они дают выигрыш. Компилятор должен соблюдать требования при подборе подходящей команды для заполнения слота задержки. Если такой команды не находится, слот задержки должен заполняться пустой операцией.

Рассматриваемый случай	Требования	Когда увеличивается производительность
(a)	Команда условного перехода не должна зависеть от переставляемой команды	Всегда
(b)	Выполнение переставляемой команды должно быть корректным, даже если переход не выполняется Может потребоваться копирование команды	Когда переход выполняется. Может увеличивать размер программы в случае копирования команды
(c)	Выполнение переставляемой команды должно быть корректным, даже если переход выполняется	Когда переход не выполняется

Рис. 5.15, б

Планирование задержанных переходов осложняется (1) наличием ограничений на команды, размещение которых планируется в слотах задержки и (2) необходимостью предсказывать во время компиляции, будет ли условный переход выполняемым или нет. Рис. 5.16 дает общее представление об эффективности планирования переходов для простейшего конвейера с одним слотом задержки перехода при использовании простого алгоритма планирования. Он показывает, что больше половины слотов задержки переходов оказываются заполненными. При этом почти 80% заполненных слотов оказываются полезными для выполнения программы. Это может показаться удивительным, поскольку условные переходы являются выполняемыми примерно в 53% случаев. Высокий процент использования заполненных слотов объясняется тем, что примерно половина из них заполняется командами, предшествовавшими команде условного перехода (стратегия (a)), выполнение которых необходимо независимо от того, выполняется ли переход, или нет.

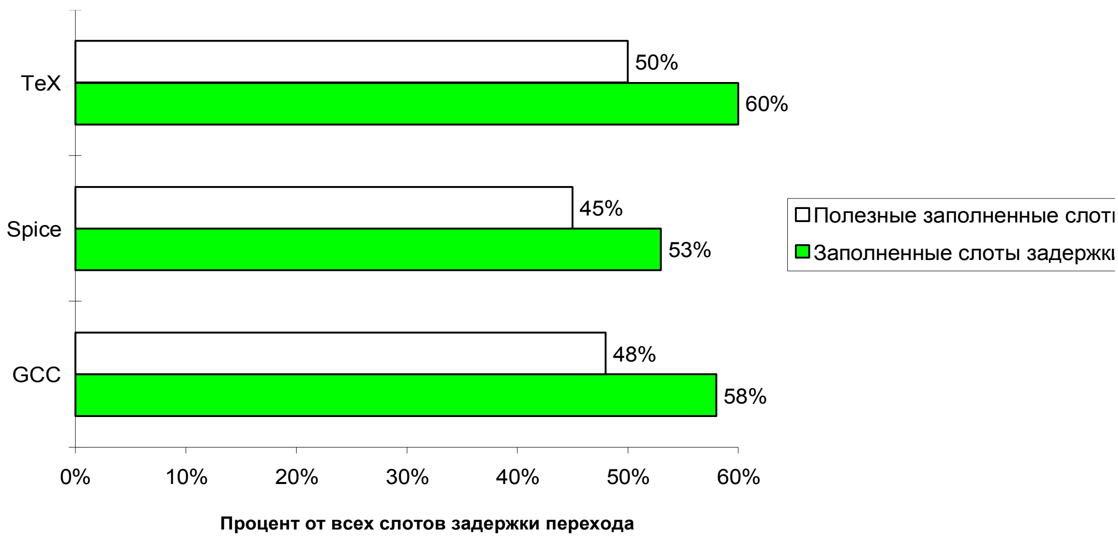


Рис. 5.16. Частота заполнения одного слота задержки условного перехода

Имеются небольшие дополнительные затраты аппаратуры на реализацию задержанных переходов. Из-за задержанного эффекта условных переходов, для корректного восстановления состояния в случае появления прерывания нужны несколько счетчиков команд (один плюс длина задержки).

Статическое прогнозирование условных переходов: использование технологии компиляторов

Имеются два основных метода, которые можно использовать для статического предсказания переходов: метод исследования структуры программы и метод использования информации о профиле выполнения программы, который собран в результате предварительных запусков программы. Использование структуры программы достаточно просто: в качестве исходной точки можно предположить, например, что все идущие назад по программе переходы являются выполняемыми, а идущие вперед по программе - невыполняемыми. Однако эта схема не очень эффективна для большинства программ. Основываясь только на структуре программы просто трудно сделать лучший прогноз.

Альтернативная техника для предсказания переходов основана на информации о профиле выполнения программы, собранной во время предыдущих прогонов. Ключевым моментом, который делает этот подход заслуживающим внимания, является то, что поведение переходов при выполнении программы часто повторяется, т.е. каждый отдельный переход в программе часто оказывается смещенным в одну из сторон: он либо выполняемый, либо невыполняемый. Проведенные многими авторами исследования показывают достаточно успешное предсказания переходов с использованием этой стратегии.

В следующей главе мы рассмотрим использование схем динамического прогнозирования, основанного на поведении программы во время ее работы. Мы также рассмотрим несколько методов планирования кода во время компиляции. Эта методика требует статического предсказания переходов, таким образом, идеи этого раздела являются важными.

4.6. **5.6. Проблемы реализации точного прерывания в конвейере**

Обработка прерываний в конвейерной машине оказывается более сложной из-за того, что совмещенное выполнение команд затрудняет определение возможности безопасного изменения состояния машины произвольной командой. В конвейерной машине команда выполняется по этапам, и ее завершение осуществляется через несколько тактов после выдачи для выполнения. Еще в процессе выполнения отдельных этапов команда может изменить состояние машины. Тем временем возникшее прерывание может вынудить машину прервать выполнение еще не завершенных команд.

Как и в неконвейерных машинах двумя основными проблемами при реализации прерываний являются: (1) прерывания возникают в процессе выполнения некоторой команды; (2) необходим механизм возврата из прерывания для продолжения выполнения программы. Например, для нашего простейшего конвейера прерывание по отсутствию страницы виртуальной памяти при выборке данных не может произойти до этапа выборки из памяти (MEM). В момент возникновения этого прерывания в процессе обработки уже будут находиться несколько команд. Поскольку подобное прерывание должно обеспечить возврат для продолжения программы и требует переключения на другой процесс (операционную систему), необходимо надежно очистить конвейер и сохранить состояние машины таким, чтобы повторное выполнение команды после возврата из прерывания осуществлялось при корректном состоянии машины. Обычно это реализуется путем сохранения адреса команды (PC), вызвавшей прерывание. Если выбранная после возврата из прерывания команда не является командой перехода, то сохраняется обычная последовательность выборки и обработки команд в конвейере. Если же это команда перехода, то мы должны оценить условие перехода и в зависимости от выбранного направления начать выборку либо по целевому адресу команды перехода, либо следующей за переходом команды. Когда происходит прерывание, для корректного сохранения состояния машины необходимо выполнить следующие шаги:

1. 1. В последовательность команд, поступающих на обработку в конвейер, принудительно вставить команду перехода на прерывание.
2. 2. Пока выполняется команда перехода на прерывание, погасить все требования записи, выставленные командой, вызвавшей прерывание, а также всеми следующими за ней в конвейере командами. Эти действия позволяют предотвратить все изменения состояния машины командами, которые не завершились к моменту начала обработки прерывания.
3. 3. После передачи управления подпрограмме обработки прерываний операционной системы, она немедленно должна сохранить значение адреса команды (PC), вызвавшей прерывание. Это значение будет использоваться позже для организации возврата из прерывания.

Если используются механизмы задержанных переходов, состояние машины уже невозможно восстановить с помощью одного счетчика команд, поскольку в процессе восстановления команды в конвейере могут оказаться вовсе не последовательными. В частности, если команда, вызвавшая прерывание, находилась в слоте задержки перехода, и переход был выполненным, то необходимо заново повторить выполнение команд из слота задержки плюс команду, находящуюся по целевому адресу команды перехода. Сама команда перехода уже выполнилась и ее повторения не требуется. При этом адреса команд из слота задержки перехода и целевой адрес команды перехода естественно не являются последовательными. Поэтому необходимо сохранять и восстанавливать несколько счетчиков команд, число которых на единицу превышает длину слота задержки. Это выполняется на третьем шаге обработки прерывания.

После обработки прерывания специальные команды осуществляют возврат из прерывания путем перезагрузки счетчиков команд и инициализации потока команд. Если конвейер может быть остановлен так, что команды, непосредственно предшествовавшие вызвавшей прерывание команде, завершаются, а следовавшие за ней могут быть заново запущены для выполнения, то говорят, что конвейер обеспечивает точное прерывание. В идеале команда, вызывающая прерывание, не должна менять состояние машины, и для корректной обработки некоторых типов прерываний требуется, чтобы команда, вызывающая прерывание, не имела никаких побочных эффектов. Для других типов прерываний, например, для прерываний по исключительным ситуациям плавающей точки, вызывающая прерывание команда на некоторых машинах записывает свои результаты еще до того момента, когда прерывание может быть обработано. В этих случаях аппаратура должна быть готовой для восстановления операндов-источников, даже если местоположение результата команды совпадает с местоположением одного из операндов-источников.

Поддержка точных прерываний во многих системах является обязательным требованием, а в некоторых системах была бы весьма желательной, поскольку она упрощает интерфейс операционной системы. Как минимум в машинах со страничной организацией памяти или с реализацией арифметической обработки в соответствии со стандартом IEEE средства обработки прерываний должны обеспечивать точное прерывание либо целиком с помощью аппаратуры, либо с помощью некоторой поддержки со стороны программных средств.

Необходимость реализации в машине точных прерываний иногда оспаривается из-за некоторых проблем, которые осложняют повторный запуск команд. Повторный запуск сложен из-за того, что команды могут изменить состояние машины еще до того, как они гарантировано завершают свое выполнение (иногда гарантированное завершение команды называется фиксацией команды или фиксацией результатов выполнения команды). Поскольку команды в конвейере могут быть взаимозависимыми, блокировка изменения состояния машины может оказаться непрактичной, если конвейер продолжает работать. Таким образом, по мере увеличения степени конвейеризации машины возникает необходимость отката любого изменения состояния, выполненного до фиксации команды. К счастью, в простых конвейерах, подобных рассмотренному, эти проблемы не возникают. На рис. 5.17 показаны ступени рассмотренного конвейера и причины прерываний, которые могут возникнуть на соответствующих ступенях при выполнении команд.

Ступень конвейера	Причина прерывания
IF	Ошибка при обращении к странице памяти при выборке команды; не выровненное обращение к памяти; нарушение защиты памяти
ID	Неопределенный или запрещенный код операции
EX	Арифметическое прерывание
MEM	Ошибка при обращении к странице памяти при выборке данных; не выровненное обращение к памяти; нарушение защиты памяти
WB	Отсутствует

Рис. 5.17. Причины прерываний в простейшем конвейере

4.7. 5.7. Обработка многотактных операций и механизмы обходов в длинных конвейерах

В рассмотренном нами конвейере стадия выполнения команды (EX) составляла всего один такт, что вполне приемлемо для целочисленных операций. Однако для большинства операций плавающей точки было бы непрактично требовать, чтобы все они выполнялись за один или даже за два такта. Это привело бы к существенному увеличению такта синхронизации конвейера, либо к сверхмерному увеличению количества оборудования (объема логических схем) для реализации устройств плавающей точки. Проще всего представить, что команды плавающей точки используют тот же самый конвейер, что и целочисленные команды, но с двумя важными изменениями. Во-первых, такт EX может повторяться многократно столько раз, сколько необходимо для выполнения операции. Во-вторых, в процессоре может быть несколько функциональных устройств, реализующих операции плавающей точки. При этом могут возникать приостановки конвейера, если выданная для выполнения команда либо вызывает структурный конфликт по функциональному устройству, которое она использует, либо существует конфликт по данным.

Допустим, что в нашей реализации процессора имеются четыре отдельных функциональных устройства:

1. 1. Основное целочисленное устройство.
2. 2. Устройство умножения целочисленных операндов и операндов с плавающей точкой.
3. 3. Устройство сложения с плавающей точкой.
4. 4. Устройство деления целочисленных операндов и операндов с плавающей точкой.

Целочисленное устройство обрабатывает все команды загрузки и записи в память при работе с двумя наборами регистров (целочисленных и с плавающей точкой), все целочисленные операции (за исключением команд умножения и деления) и все команды переходов. Если предположить, что стадии выполнения других функциональных устройств неконвейерные, то рис. 5.18 показывает структуру такого конвейера. Поскольку стадия EX является неконвейерной, никакая команда, использующая функциональное устройство, не может быть выдана для выполнения до тех пор, пока предыдущая команда не покинет ступень EX. Более того, если команда не может поступить на ступень EX, весь конвейер за этой командой будет приостановлен.

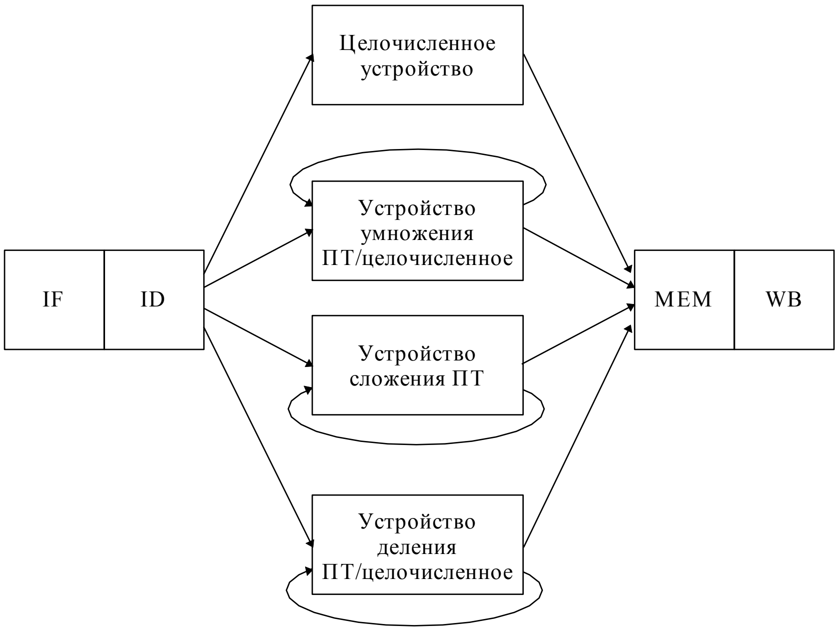


Рис. 5.18. Конвейер с дополнительными функциональными устройствами

В действительности промежуточные результаты, возможно, не используются циклически ступенью EX, как это показано на рис. 5.18, и ступень EX имеет задержки длительностью более одного такта. Мы можем обобщить структуру конвейера плавающей точки, допустив конвейеризацию некоторых ступеней и параллельное выполнение нескольких операций. Чтобы описать работу такого конвейера, мы должны определить задержки функциональных устройств, а также скорость инициаций или скорость повторения операций. Это скорость, с которой новые операции данного типа могут поступать в функциональное устройство. Например, предположим, что имеют место следующие задержки функциональных устройств и скорости повторения операций:

Функциональное устройство	Задержка	Скорость повторения
Целочисленное АЛУ	1	1
Сложение с ПТ	4	2
Умножение с ПТ (и целочисленное)	6	3
Деление с ПТ (и целочисленное)	15	15

На рис. 5.19 представлена структура подобного конвейера. Ее реализация требует введения конвейерной регистровой станции EX1/EX2 и модификации связей между регистрами ID/EX и EX/MEM.

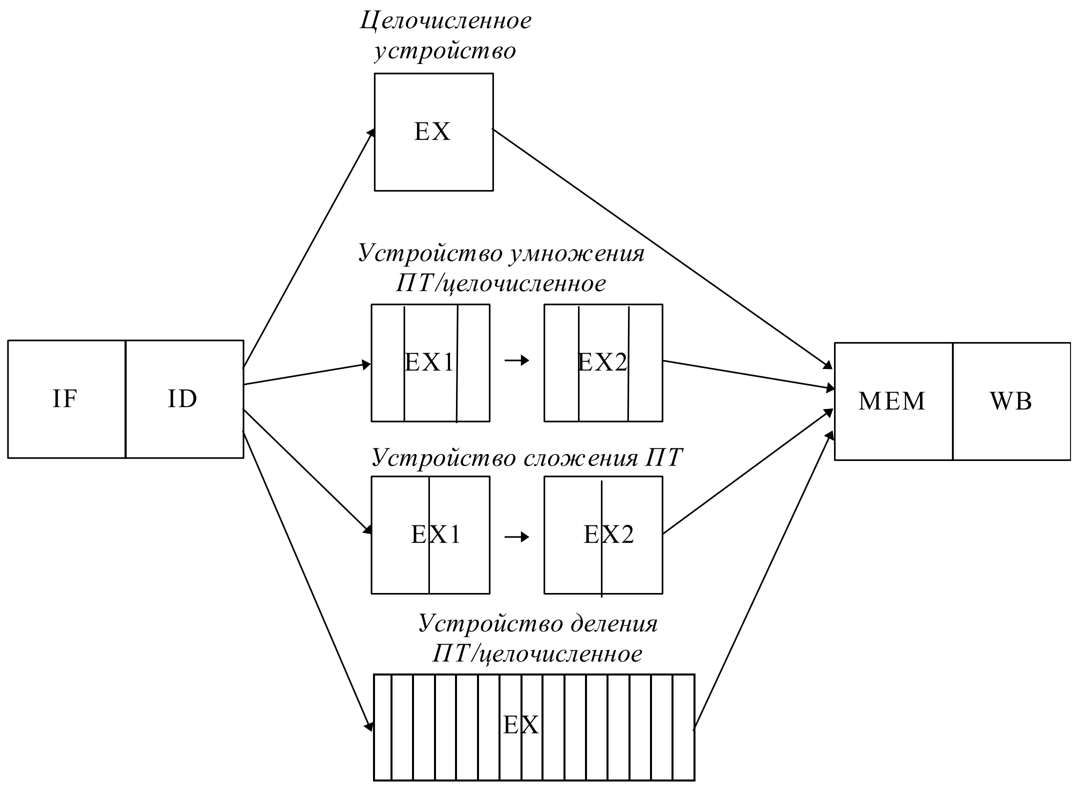


Рис. 5.19. Конвейер с многоступенчатыми функциональными устройствами

Конфликты и ускоренные пересылки в длинных конвейерах

Имеется несколько различных аспектов обнаружения конфликтов и организации ускоренной пересылки данных в конвейерах, подобных представленному на рис. 5.19:

1. 1. Поскольку устройства не являются полностью конвейерными, в данной схеме возможны структурные конфликты. Эти ситуации необходимо обнаруживать и приостанавливать выдачу команд.
2. 2. Поскольку устройства имеют разные времена выполнения, количество записей в регистровый файл в каждом такте может быть больше 1.
3. 3. Возможны конфликты типа WAW, поскольку команды больше не поступают на ступень WB в порядке их выдачи для выполнения. Заметим, что конфликты типа WAR невозможны, поскольку чтение регистров всегда осуществляется на ступени ID.
4. 4. Команды могут завершаться не в том порядке, в котором они были выданы для выполнения, что вызывает проблемы с реализацией прерываний.

Прежде чем представить общее решение для реализации схем обнаружения конфликтов, рассмотрим вторую и третью проблемы.

Если предположить, что файл регистров с ПТ имеет только один порт записи, то последовательность операций с ПТ, а также операция загрузки ПТ совместно с операциями ПТ может вызвать конфликты по порту записи в регистровый файл. Рассмотрим последовательность команд, представленную на рис. 5.20. В такте 10 все три команды достигнут ступени WB и должны произвести запись в регистровый файл. При наличии только одного порта записи в регистровый файл машина должна обеспечить последовательное завершение команд. Этот единственный регистровый порт является источником структурных конфликтов. Чтобы решить эту проблему, можно увеличить количество портов в регистровом файле, но такое решение может оказаться неприемлемым, поскольку эти дополнительные порты записи, скорее всего, будут редко использоваться. Однако в установившемся состоянии максимальное количество необходимых портов записи равно 1. Поэтому в реальных машинах разработчики предпочитают отслеживать обращения к порту записи в регистры и рассматривать одновременное к нему обращение как структурный конфликт.

Команда	**Номер такта**									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
`MULTD F0,F4,F6`	IF	ID	$EX1_1$	$EX1_2$	$EX1_3$	$EX2_1$	$EX2_2$	$EX2_3$	MEM	WB
`...`		IF	ID	EX	MEM	WB				
`ADDD F2,F4,F6`			IF	ID	$EX1_1$	$EX1_2$	$EX2_1$	$EX2_2$	MEM	WB
`...`				IF	ID	EX	MEM	WB		
`...`					IF	ID	EX	MEM	WB	
`LD F8,0(R2)`						IF	ID	EX	MEM	WB

Рис. 5.20. Пример конфликта по записи в регистровый файл

Имеется два способа для обхода этого конфликта. Первый заключается в отслеживании использования порта записи на ступени ID конвейера и приостановке выдачи команды как при структурном конфликте. Схема обнаружения такого конфликта обычно реализуется с помощью сдвигового регистра. Альтернативная схема предполагает приостановку конфликтующей команды, когда она пытается попасть на ступень MEM конвейера. Преимуществом такой схемы является то, что она не требует обнаружения конфликта до входа на ступень MEM, где это легче сделать. Однако подобная реализация усложняет управление конвейером, поскольку приостановки в этом случае могут возникать в двух разных местах конвейера.

Другой проблемой является возможность конфликтов типа WAW. Можно рассмотреть тот же пример, что и на рис. 5.20. Если бы команда LD была выдана на один такт раньше и имела в качестве месторасположения результата регистр F2, то возник бы конфликт типа WAW, поскольку эта команда выполняла бы запись в регистр F2 на один такт раньше команды ADDD. Имеются два способа обработки этого конфликта типа WAW. Первый подход заключается в задержке выдачи команды загрузки до момента передачи команды ADDD на ступень MEM. Второй подход заключается в подавлении результата операции сложения при обнаружении конфликта и изменении управления таким образом, чтобы команда сложения не записывала свой результат. Тогда команда LD может выдаваться для выполнения сразу же. Поскольку такой конфликт является редким, обе схемы будут работать достаточно хорошо. В любом случае конфликт может быть обнаружен на ранней стадии ID, когда команда LD выдается для выполнения. Тогда приостановка команды LD или установка блокировки записи результата командой ADDD реализуются достаточно просто.

Таким образом, для обнаружения возможных конфликтов необходимо рассматривать конфликты между командами ПТ, а также конфликты между командами ПТ и целочисленными командами. За исключением команд загрузки/записи с ПТ и команд пересылки данных между регистрами ПТ и целочисленными регистрами, команды ПТ и целочисленные команды достаточно хорошо разделены, и все целочисленные команды работают с целочисленными регистрами, а команды ПТ - с регистрами ПТ.

Таким образом, для обнаружения конфликтов между целочисленными командами и командами ПТ необходимо рассматривать только команды загрузки/записи с ПТ и команды пересылки регистров ПТ. Это упрощение управления конвейером является дополнительным преимуществом поддержания отдельных регистровых файлов для хранения целочисленных данных и данных с ПТ. (Главное преимущество заключается в удвоении общего количества регистров и увеличении пропускной способности без увеличения числа портов в каждом наборе). Если предположить, что конвейер выполняет обнаружение всех конфликтов на стадии ID, перед выдачей команды для выполнения в функциональные устройства должны быть выполнены три проверки:

1. 1. *Проверка наличия структурных конфликтов.* Ожидание освобождения функционального устройства и порта записи в регистры, если он потребуется.
2. 2. *Проверка наличия конфликтов по данным типа RAW.* Ожидание до тех пор, пока регистры-источники операндов указаны в качестве регистров результата на конвейерных станциях ID/EX (которые соответствуют команде, выданной в предыдущем такте), EX1/EX2 или EX/MEM.
3. 3. *Проверка наличия конфликтов типа WAW.* Проверка того, что команды, находящиеся на конвейерных станциях EX1 и EX2, не имеют в качестве месторасположения результата регистр результата выдаваемой для выполнения команды. В противном случае выдача команды, находящейся на ступени ID, приостанавливается.

Хотя логика обнаружения конфликтов для многотактных операций ПТ несколько более сложная, концептуально она не отличается от такой же логики для целочисленного конвейера. То же самое касается логики для ускоренной пересылки данных. Логика ускоренной пересылки данных может быть реализована с помощью проверки того, что указанный на конвейерных станциях EX/MEM и MEM/WB регистр результата является регистром операнда команды ПТ. Если происходит такое совпадение, для пересылки данных разрешается прием по соответствующему входу мультиплексора. Многотактные операции ПТ создают также новые проблемы для механизма прерывания.

Поддержка точных прерываний

Другая проблема, связанная с реализацией команд с большим временем выполнения, может быть проиллюстрирована с помощью следующей последовательности команд:

```
DIVF F0,F2,F4
ADDF F10,F10,F8
SUBF F12,F12,F14
```

Эта последовательность команд выглядит очень просто. В ней отсутствуют какие-либо зависимости. Однако она приводит к появлению новых проблем из-за того, что выданная раньше команда может завершиться после команды, выданной для выполнения позже. В данном примере можно ожидать, что команды ADDF и SUBF завершаться раньше, чем завершится команда DIVF. Этот эффект является типичным для конвейеров команд с большим временем выполнения и называется *внеочередным завершением команд (out-of-order completion)*. Тогда, например, если команда DIVF вызовет арифметическое прерывание после завершения команды ADDF, мы не сможем реализовать точное прерывание на уровне аппаратуры. В действительности, поскольку команда ADDF меняет значение одного из своих операндов, невозможно даже с помощью программных средств восстановить состояние, которое было перед выполнением команды DIVF.

Имеются четыре возможных подхода для работы в условиях внеочередного завершения команд. Первый из них просто игнорирует проблему и предлагает механизмы неточного прерывания. Этот подход использовался в 60-х и 70-х годах и все еще применяется в некоторых суперкомпьютерах, в которых некоторые классы прерываний запрещены или обрабатываются аппаратурой без остановки конвейера. Такой подход трудно использовать в современных машинах при наличии концепции виртуальной памяти и стандарта на операции с плавающей точкой IEEE, которые требуют реализации точного прерывания путем комбинации аппаратных и программных средств. В некоторых машинах эта проблема решается путем введения двух режимов выполнения команд: быстрого, но с возможно не точными прерываниями, и медленного, гарантирующего реализацию точных прерываний.

Второй подход заключается в буферизации результатов операции до момента завершения выполнения всех команд, предшествовавших данной. В некоторых машинах используется этот подход, но он становится все более дорогостоящим, если отличия во времени выполнения разных команд велики, поскольку становится большим количество результатов, которые необходимо буферизовать. Более того, результаты из этой буферизованной очереди необходимо пересылать для обеспечения продолжения выдачи новых команд. Это требует большого количества схем сравнения и многовходовых мультиплексоров. Имеются две вариации этого основного подхода. Первая называется буфером истории (history file), который использовался в машине CYBER 180/990. Буфер истории отслеживает первоначальные значения регистров. Если возникает прерывание и состояние машины необходимо откатить назад до точки, предшествовавшей некоторым завершившимся вне очереди командам, то первоначальное значение регистров может быть восстановлено из этого буфера истории. Подобная методика использовалась также при реализации автоинкрементной и автодекрементной адресации в машинах типа VAX. Другой подход называется буфером будущего (future file). Этот буфер хранит новые значения регистров. Когда все предшествующие команды завершены, основной регистровый файл обновляется значениями из этого буфера. При прерывании основной регистровый файл хранит точные значения регистров, что упрощает организацию прерывания. В следующей главе будут рассмотрены некоторые расширения этой идеи.

Третий используемый метод заключается в том, чтобы разрешить в ряде случаев неточные прерывания, но при этом сохранить достаточно информации, чтобы подпрограмма обработки прерывания могла выполнить точную последовательность прерывания. Это предполагает наличие информации о находившихся в конвейере командах и их адресов. Тогда после обработки прерывания, программное обеспечение завершает выполнение всех команд, предшествовавших последней завершившейся команде, а затем последовательность может быть запущена заново. Рассмотрим следующий наихудший случай:

Команда 1 - длинная команда, которая, в конце концов, вызывает прерывание

Команда 2, ... , Команда n-1 - последовательность команд, выполнение которых не завершилось

Команда n - команда, выполнение которой завершилось

Имея значения адресов всех команд в конвейере и адрес возврата из прерывания, программное обеспечение может определить состояние команды 1 и команды n. Поскольку команда n завершила выполнение, хотелось бы продолжить выполнение с команды n+1. После обработки прерывания программное обеспечение должно смоделировать выполнение команд с 1 по n-1. Тогда можно осуществить возврат из прерывания на команду n+1. Наибольшая неприятность такого подхода связана с усложнением подпрограммы обработки прерывания. Но для простых конвейеров, подобных рассмотренному нами, имеются и упрощения. Если команды с 2 по n все являются целочисленными, то мы просто знаем, что в случае завершения выполнения команды n, все команды с 2 по n-1 также завершили выполнение. Таким образом, необходимо обрабатывать только операцию с плавающей точкой. Чтобы сделать эту схему работающей, количество операций ПТ, выполняющихся с совмещением, может быть ограничено. Например, если допускается совмещение только двух операций, то только прерванная команда должна завершаться программными средствами. Это ограничение может снизить потенциальную пропускную способность, если конвейеры плавающей точки являются достаточно длинными или если имеется значительное количество функциональных устройств. Такой подход использовался в архитектуре SPARC, позволяющей совмещать выполнение целочисленных операций с операциями плавающей точки.

Четвертый метод представляет собой гибридную схему, которая позволяет продолжать выдачу команд, только если известно, что все команды, предшествовавшие выдаваемой, будут завершены без прерывания. Это гарантирует, что в случае возникновения прерывания ни одна следующая за ней команда не будет завершена, а все предшествующие будут завершены. Иногда это означает необходимость приостановки машины для поддержки точных прерываний. Чтобы эта схема работала, необходимо, чтобы функциональные устройства плавающей точки определяли возможность появления прерывания на самой ранней стадии выполнения команд так, чтобы предотвратить завершение выполнения следующих команд. Такая схема используется, например, в микропроцессорах R2000/R3000 и R4000 компании MIPS.

5. 6. Конвейерная и суперскалярная обработка

5.1. 6.1. Параллелизм на уровне выполнения команд, планирование загрузки конвейера и методика разворачивания циклов

В предыдущей главе мы рассмотрели средства конвейеризации, которые обеспечивают совмещенный режим выполнения команд, когда они являются независимыми друг от друга. Это потенциальное совмещение выполнения команд называется параллелизмом на уровне команд. В данной главе мы рассмотрим ряд методов развития идей конвейеризации, основанных на увеличении степени параллелизма, используемой при выполнении команд. Мы начнем с рассмотрения методов, позволяющих снизить влияние конфликтов по данным и по управлению, а затем вернемся к теме расширения возможностей процессора по использованию параллелизма, заложенного в программах. Затем мы обсудим современные технологии компиляторов, используемые для увеличения степени параллелизма уровня команд.

Для начала запишем выражение, определяющее среднее количество тактов для выполнения команды в конвейере:

CPI конвейера = CPI идеального конвейера +

+ Приостановки из-за структурных конфликтов +

+ Приостановки из-за конфликтов типа RAW +

+ Приостановки из-за конфликтов типа WAR +

+ Приостановки из-за конфликтов типа WAW +

+ Приостановки из-за конфликтов по управлению

CPI идеального конвейера есть не что иное, как максимальная пропускная способность, достижимая при реализации. Уменьшая каждое из слагаемых в правой части выражения, мы минимизируем общий CPI конвейера и таким образом увеличиваем пропускную способность команд. Это выражение позволяет также охарактеризовать различные методы, которые будут рассмотрены в этой главе, по тому компоненту общего CPI, который соответствующий метод уменьшает. На рис. 6.1 показаны некоторые методы, которые будут рассмотрены, и их воздействие на величину CPI.

Прежде, чем начать рассмотрение этих методов, необходимо определить концепции, на которых эти методы построены.

Параллелизм уровня команд: зависимости и конфликты по данным

Все рассматриваемые в этой главе методы используют параллелизм, заложенный в последовательности команд. Как мы установили выше, этот тип параллелизма называется параллелизмом уровня команд или ILP. Степень параллелизма, доступная внутри базового блока (линейной последовательности команд, переходы из вне которой разрешены только на ее вход, а переходы внутри которой разрешены только на ее выход), достаточно мала. Например, средняя частота переходов в целочисленных программах составляет около 16%. Это означает, что в среднем между двумя переходами выполняются примерно пять команд. Поскольку эти пять команд возможно взаимозависимые, то степень перекрытия, которую мы можем использовать внутри базового блока, возможно, будет меньше чем пять. Чтобы получить существенное улучшение производительности, мы должны использовать параллелизм уровня команд одновременно для нескольких базовых блоков.

Метод	Снижает
Разворачивание циклов	Приостановки по управлению
Базовое планирование конвейера	Приостановки RAW
Динамическое планирование с централизованной схемой управления	Приостановки RAW
Динамическое планирование с переименованием регистров	Приостановки WAR и WAW
Динамическое прогнозирование переходов	Приостановки по управлению
Выдача нескольких команд в одном такте	Идеальный CPI
Анализ зависимостей компилятором	Идеальный CPI и приостановки по данным
Программная конвейеризация и планирование трасс	Идеальный CPI и приостановки по данным
Выполнение по предположению	Все приостановки по данным и управлению
Динамическое устранение неоднозначности памяти	Приостановки RAW, связанные с памятью

Рис. 6.1.

Самый простой и общий способ увеличения степени параллелизма, доступного на уровне команд, является использование параллелизма между итерациями цикла. Этот тип параллелизма часто называется параллелизмом уровня итеративного цикла. Ниже приведен простой пример цикла, выполняющего сложение двух 1000-элементных векторов, который является полностью параллельным:

```
for (i = 1; i <= 1000; i = i + 1)
x[i] = x[i] + y[i];
```

Каждая итерация цикла может перекрываться с любой другой итерацией, хотя внутри каждой итерации цикла практическая возможность перекрытия небольшая.

Имеется несколько методов для превращения такого параллелизма уровня цикла в параллелизм уровня команд. Эти методы основаны главным образом на разворачивании цикла либо статически с помощью компилятора, либо динамически с помощью аппаратуры. Ниже в этом разделе мы рассмотрим подробный пример разворачивания цикла.

Важным альтернативным методом использования параллелизма уровня команд является использование векторных команд. По существу векторная команда оперирует с последовательностью элементов данных. Например, приведенная выше последовательность на типичной векторной машине может быть выполнена с помощью четырех команд: двух команд загрузки векторов x и y из памяти, одной команды сложения двух векторов и одной команды записи вектора-результата. Конечно, эти команды могут быть конвейеризованными и иметь относительно большие задержки выполнения, но эти задержки могут перекрываться. Векторные команды и векторные машины заслуживают отдельного рассмотрения, которое выходит за рамки данного курса. Хотя разработка идей векторной обработки предшествовала появлению большинства методов использования параллелизма, которые рассматриваются в этой главе, машины, использующие параллелизм уровня команд постепенно заменяют машины, базирующиеся на векторной обработке. Причины этого сдвига технологии обсуждаются более детально позже в данном курсе.

Зависимости

Чтобы точно определить, что мы понимаем под параллелизмом уровня цикла и параллелизмом уровня команд, а также для количественного определения степени доступного параллелизма, мы должны определить, что такое параллельные команды и параллельные циклы. Начнем с объяснения того, что такое пара параллельных команд. Две команды являются параллельными, если они могут выполняться в конвейере одновременно без приостановок, предполагая, что конвейер имеет достаточно ресурсов (структурные конфликты отсутствуют).

Поэтому, если между двумя командами существует взаимозависимость, то они не являются параллельными. Имеется три типа зависимостей: зависимости по данным, зависимости по именам и зависимости по управлению. Команда j *зависит по данным* от команды i, если имеет место любое из следующих условий:

- • команда i вырабатывает результат, который использует команда j
- • команда j является зависимой по данным от команды k, а команда k является зависимой по данным от команды i

Второе условие просто означает, что одна команда зависит от другой, если между этими двумя командами имеется цепочка зависимостей первого типа. Эта цепочка зависимостей может быть длиною во всю программу.

Если две команды являются зависимыми по данным, они не могут выполняться одновременно или полностью совмещено. Зависимость по данным предполагает, что между двумя командами имеется цепочка из одного или нескольких конфликтов типа RAW. Одновременное выполнение таких команд требует создания машины с внутренними схемами блокировок конвейера, обеспечивающих обнаружение конфликтов и уменьшение времени приостановок или полное устранение перекрытия. В машине без внутренних блокировок, которые базируются на программном планировании работы конвейера компилятором, компилятор не может спланировать зависимые команды так, чтобы они полностью совмещались, поскольку в противном случае программа не будет выполняться правильно. Наличие зависимостей по данным в последовательности команд отражает зависимость по данным в исходном тексте программы, на основании которого она генерировалась. Эффект первоначальной зависимости по данным должен сохраняться.

Зависимости являются свойством программ. Приведет ли данная зависимость к обнаруживаемому конфликту и вызовет ли данный конфликт реальную приостановку конвейера, зависит от организации конвейера. Действительно, многие методы, рассматриваемые в этой главе, позволяют обойти конфликты или обойти необходимость приостановки конвейера в случае возникновения конфликта, при сохранении зависимости. Важность зависимостей по данным заключается в том, что именно они устанавливают верхнюю границу степени параллелизма, который вероятно может быть использован. Наличие зависимостей по данным означает также, что результаты должны вычисляться в определенном порядке, поскольку более поздняя команда зависит от результата предыдущей команды.

Данные могут передаваться от команды к команде либо через регистры, либо через ячейки памяти. Когда данные передаются через регистры, обнаружение зависимостей значительно упрощается, поскольку имена регистров зафиксированы в командах (хотя этот процесс становится более сложным, если вмешиваются условные переходы). Зависимости по данным, которые передаются через ячейки памяти, обнаружить значительно сложнее, поскольку два адреса могут относиться к одной и той же ячейке памяти, но внешне выглядят по-разному (например, 100(R4) и 20(R6) могут определять один и тот же адрес). Кроме того, эффективный адрес команды загрузки или записи может меняться от одного выполнения команды к другому (так что 20(R4) и 20(R4) будут определять разные адреса), еще больше усложняя обнаружение зависимости. В этой главе мы рассмотрим как аппаратные, так и программные методы обнаружения зависимостей по данным, которые связаны с ячейками памяти. Методы компиляции для обнаружения таких зависимостей являются очень важными при выявлении параллелизма уровня цикла.

Вторым типом зависимостей в программах являются *зависимости по именам*. Зависимости по именам возникают, когда две команды используют одно и то же имя (либо регистра, либо ячейки памяти), но при отсутствии передачи данных между командами. Имеется два типа зависимости имен между командой i, которая предшествует команде j в программе:

1. 1. *Антизависимость* между командой i и командой j возникает тогда, когда команда j записывает в регистр или ячейку памяти, который(ую) команда i считывает и команда i выполняется первой. Антизависимость соответствует конфликту типа WAR, и обнаружение конфликтов типа WAR означает упорядочивание выполнения пары команд с антизависимостью.
2. 2. *Зависимость по выходу* возникает, когда команда i и команда j записывают результат в один и тот же регистр или в одну и ту же ячейку памяти. Порядок выполнения этих команд должен сохраняться. Зависимости по выходу сохраняются путем обнаружения конфликтов типа WAW.

Как антизависимости, так и зависимости по выходу являются зависимостями по именам, в отличие от истинных зависимостей по данным, поскольку в них отсутствует передача данных от одной команды к другой. Это означает, что команды, связанные зависимостью по именам, могут выполняться одновременно или могут быть переупорядочены, если имя (номер регистра или адрес ячейки памяти), используемое в командах изменяется так, что команды не конфликтуют. Это переименование может быть выполнено более просто для регистровых операндов и называется переименованием регистров (register renaming). Переименование регистров может выполняться либо статически компилятором, или динамически аппаратными средствами.

В качестве примера рассмотрим следующую последовательность команд:

```
ADD R1,R2,R3
SUB R2,R3,R4
AND R5,R1,R2
OR  R1,R3,R4
```

В этой последовательности имеется антизависимость по регистру R2 между командами ADD и SUB, которая может привести к конфликту типа WAR. Ее можно устранить путем переименования регистра результата команды SUB, например, на R6 и изменения всех последующих команд, которые используют результат команды вычитания, для использования этого регистра R6 (в данном случае это только последний операнд в команде AND). Использование R1 в команде OR приводит как к зависимости по выходу с командой ADD, так и к антизависимости между командами OR и AND. Обе зависимости могут быть устранены путем замены регистра результата либо команды ADD, либо команды OR. В первом случае должна измениться каждая команда, которая использует результат команды ADD, прежде чем команда OR запишет в регистр R1 (а именно, второй операнд команды AND в данном примере). Во втором случае при замене регистра результата команды OR, все последующие команды, использующие ее результат, должны также измениться. Альтернативой переименованию в процессе компиляции является аппаратное переименование регистров, которое может быть использовано в ситуациях, когда возникают условные переходы, которые возможно сложны или невозможны для анализа компилятором; в следующем разделе эта методика обсуждается более подробно.

Последним типом зависимостей являются *зависимости по управлению*. Зависимости по управлению определяют порядок команд по отношению к команде условного перехода так, что команды, не являющиеся командами перехода, выполняются только когда они должны выполняться. Каждая команда в программе является зависимой по управлению от некоторого набора условных переходов и, в общем случае, эти зависимости по управлению должны сохраняться. Одним из наиболее простых примеров зависимости по управлению является зависимость операторов, находящихся в части "then" оператора условного перехода if. Например, в последовательности кода:

```
if p1 {
   S1;
     };
if p2 {
   S2;
     }
```

S1 является зависимым по управлению от p1, а S2 зависит по управлению от p2 и не зависит от p1.

Имеются два ограничения, связанные с зависимостями по управлению:

1. 1. Команда, которая зависит по управлению от условного перехода, не может быть в результате перемещения поставлена перед командой условного перехода так, что ее выполнение более не управлялось бы этим условным переходом. Например, мы не можем взять команду из части "then" оператора if и поставить ее перед оператором if.
2. 2. Команда, которая не является зависимой по управлению от команды условного перехода, не может быть поставлена после команды условного перехода так, что ее выполнение станет управляться этим условным переходом. Например, мы не можем взять оператор, стоящий перед оператором if и перенести его в часть "then" условного оператора.

Следующий пример иллюстрирует эти два ограничения:

```
            ADD   R1,R2,R3
            BEQZ  R12,skipnext
            SUB   R4,R5,R6
skipnext:   OR    R7,R1,R9
            MULT  R13,R1,R4
```

В этой последовательности команд имеются следующие зависимости по управлению (предполагается, что переходы не задерживаются). Команда SUB зависит по управлению от команды BEQZ, поскольку изменение порядка следования этих команд изменит и результат вычислений. Если поставить команду SUB перед командой условного перехода, результат команды MULT не будет тем же самым, что и в случае, когда условный переход является выполняемым. Аналогично, команда ADD не может быть поставлена после команды условного перехода, поскольку это приведет к изменению результата команды MULT, когда переход является выполняемым. Команда OR не является зависимой по управлению от условного перехода, поскольку она выполняется независимо от того, является ли переход выполняемым или нет. Поскольку команда OR не зависит по данным от предшествующих команд и не зависит по управлению от команды условного перехода, она может быть поставлена перед командой условного перехода, что не приведет к изменению значения, вычисляемого этой последовательностью команд. Конечно это предполагает, что команда OR не вызывает никаких побочных эффектов (например, возникновения исключительной ситуации). Если мы хотим переупорядочить команды с подобными потенциальными побочными эффектами, требуется дополнительный анализ компилятором или дополнительная аппаратная поддержка.

Обычно зависимости по управлению сохраняются посредством двух свойств простых конвейеров, подобных рассмотренным в предыдущей главе. Во-первых, команды выполняются в порядке, предписанном программой. Это гарантирует, что команда, стоящая перед командой условного перехода, выполняется перед переходом; таким образом, команда ADD в выше приведенной последовательности будет выполняться перед условным переходом. Во-вторых, средства обнаружения конфликтов по управлению или конфликтов условных переходов гарантируют, что команда, зависимая по управлению от условного перехода, не будет выполняться до тех пор, пока не известно направление условного перехода. В частности команда SUB не будет выполняться до тех пор, пока машина не определит, что условный переход является невыполняемым.

Хотя сохранение зависимостей по управлению является полезным и простым способом обеспечения корректности программы, сама по себе зависимость по управлению не является фундаментальным ограничением производительности. Возможно, мы были бы рады выполнять команды, которые не должны выполняться, тем самым нарушая зависимости по управлению, если бы могли это делать, не нарушая корректность программы. Зависимость по управлению не является критическим свойством, которое должно сохраняться. В действительности, двумя свойствами, которые являются критичными с точки зрения корректности программы и которые обычно сохраняются посредством зависимостей по управлению, являются поведение исключительных ситуаций (exception behavior) и поток данных (data flow).

Сохранение поведения исключительных ситуаций означает, что любые изменения в порядке выполнения команд не должны менять условия возникновения исключительных ситуаций в программе. Часто это требование можно смягчить: переупорядочивание выполнения команд не должно приводить к появлению в программе новых исключительных ситуаций. Простой пример показывает, как поддержка зависимостей по управлению может сохранить эти ситуации. Рассмотрим кодовую последовательность:

```
BEQZ  R2,L1
LW    R1,0(R2)
L1:
```

В данном случае, если мы игнорируем зависимость по управлению и ставим команду загрузки перед командой условного перехода, команда загрузки может вызвать исключительную ситуацию по защите памяти. Заметим, что здесь зависимость по данным, которая препятствует перестановке команд BEQZ и LW, отсутствует, это только зависимость по управлению. Подобная ситуация может возникнуть и при выполнении операции с ПТ, которая может вызвать исключительную ситуацию. В любом случае, если переход выполняется, то исключительная ситуация не возникнет, если команда не ставится выше команды условного перехода. Чтобы разрешить переупорядочивание команд мы хотели бы как раз игнорировать исключительную ситуацию, если переход не выполняется. В разд. 6.7 мы рассмотрим два метода, выполнение по предположению и условные команды, которые позволяют нам справиться с этой проблемой.

Вторым свойством, сохраняемым с помощью поддержки зависимостей по управлению, является поток данных. Поток данных представляет собой действительный поток данных между командами, которые вырабатывают результаты, и командами, которые эти результаты используют. Условные переходы делают поток данных динамическим, поскольку они позволяют данным для конкретной команды поступать из многих точек (источников). Рассмотрим следующую последовательность команд:

```
ADD   R1,R2,R3
BEQZ  R4,L
SUB   R1,R5,R6
```

```
L:  OR    R7,R1,R8
```

В этом примере значение R1, используемое командой OR, зависит от того, выполняется или не выполняется условный переход. Одной зависимости по данным не достаточно для сохранения корректности программы, поскольку она имеет дело только со статическим порядком чтения и записи. Таким образом, хотя команда OR зависит по данным как от команды ADD, так и от команды SUB, этого недостаточно для корректного выполнения. Когда выполняются команды, должен сохраняться поток данных: если переход не выполняется, то команда OR должна использовать значение R1, вычисленное командой SUB, а если переход выполняется - значение R1, вычисленное командой ADD. Перестановка команды SUB на место перед командой условного перехода не меняет статической зависимости, но она определенно повлияет на поток данных и таким образом приведет к некорректному выполнению. При сохранении зависимости по управлению команды SUB от условного перехода, мы предотвращаем незаконное изменение потока данных. Выполнение команд по предположению и условные команды, которые помогают решить проблему исключительных ситуаций, позволяют также изменить зависимость по управлению, поддерживая при этом правильный поток данных (разд. 6.7).

Иногда мы можем определить, что устранение зависимости по управлению, не может повлиять на поведение исключительных ситуаций, либо на поток данных. Рассмотрим слегка модифицированную последовательность команд:

```
           ADD   R1,R2,R3
           BEQZ  R12,skipnext
           SUB   R4,R5,R6
           ADD   R5,R4,R9
skipnext:  OR    R7,R8,R9
```

Предположим, что мы знаем, что регистр результата команды SUB (R4) не используется после команды, помеченной меткой skipnext. (Свойство, определяющее, будет ли значение использоваться последующими командами, называется живучестью (liveness) и мы вскоре определим его более формально). Если бы регистр R4 не использовался, то изменение значения R4 прямо перед выполнением условного перехода не повлияло бы на поток данных. Таким образом, если бы регистр R4 не использовался, и команда SUB не могла выработать исключительную ситуацию, мы могли бы поместить команду SUB на место перед командой условного перехода, поскольку на результат программы это изменение не влияет. Если переход выполняется, команда SUB выполнится и будет бесполезна, но она не повлияет на результат программы. Этот тип планирования кода иногда называется планированием по предположению (speculation), поскольку компилятор в основном делает ставку на исход условного перехода; в данном случае предполагается, что условный переход обычно является невыполняемым. Более амбициозные механизмы планирования по предположению в компиляторах обсуждаются в разд. 6.7.

Механизмы задержанных переходов, которые мы рассматривали в предыдущей главе, могут использоваться для уменьшения простоев, возникающих по вине условных переходов, и иногда позволяют использовать планирование по предположению для оптимизации задержек переходов.

Зависимости по управлению сохраняются путем реализации схемы обнаружения конфликта по управлению, которая приводит к приостановке конвейера по управлению. Приостановки по управлению могут устраняться или уменьшаться множеством аппаратных и программных методов. Например, задержанные переходы могут уменьшать приостановки, возникающие в результате конфликтов по управлению. Другие методы уменьшения приостановок, вызванных конфликтами по управлению, включают разворачивание циклов, преобразование условных переходов в условно выполняемые команды и планирование по предположению, выполняемое с помощью компилятора или аппаратуры. В данной главе будут рассмотрены большинство этих методов.

Параллелизм уровня цикла: концепции и методы

Параллелизм уровня цикла обычно анализируется на уровне исходного текста программы или близкого к нему, в то время как анализ параллелизма уровня команд главным образом выполняется, когда команды уже сгенерированы компилятором. Анализ на уровне циклов включает определение того, какие зависимости существуют между операндами в цикле в пределах одной итерации цикла. Теперь мы будем рассматривать только зависимости по данным, которые возникают, когда операнд записывается в некоторой точке и считывается в некоторой более поздней точке. Мы обсудим коротко зависимости по именам. Анализ параллелизма уровня цикла фокусируется на определении того, зависят ли по данным обращения к данным в последующей итерации от значений данных, вырабатываемых в более ранней итерации.

Рассмотрим следующий цикл:

```
for (i=1; i<=100; i=i+1)  {
 A[i+1] = A[i] + C[i]; /* S1 */
 B[i+1] = B[i] + A[i+1];} /*S2*/
}
```

Предположим, что A, B и C представляют собой отдельные, неперекрывающиеся массивы. На практике иногда массивы могут быть теми же самыми или перекрываться. Поскольку массивы могут передаваться в качестве параметров некоторой процедуре, которая содержит этот цикл, определение того, перекрываются ли массивы или они совпадают, требует изощренного, межпроцедурного анализа программы. Какие зависимости по данным имеют место между операторами этого цикла?

Имеются две различных зависимости:

1. 1. S1 использует значение, вычисляемое оператором S1 на более ранней итерации, поскольку итерация i вычисляет A[i+1], которое считывается в итерации i+1. То же самое справедливо для оператора S2 для B[i] и B[i+1].
2. 2. S2 использует значение A[i+1], вычисляемое оператором S1 в той же самой итерации.

Эти две зависимости отличаются друг от друга и производят различный эффект. Чтобы увидеть, чем они отличаются, предположим, что в каждый момент времени существует только одна из этих зависимостей. Рассмотрим зависимость оператора S1 от более ранней итерации S1. Эта зависимость (loop-carried dependence) означает, что между различными итерациями цикла существует зависимость по данным. Более того, поскольку оператор S1 зависит от самого себя, последовательные итерации оператора S1 должны выполняться упорядочено.

Вторая зависимость (S2 зависит от S1) не передается от итерации к итерации. Таким образом, если бы это была единственная зависимость, несколько итераций цикла могли бы выполняться параллельно, при условии, что каждая пара операторов в итерации поддерживается в заданном порядке.

Имеется третий тип зависимостей по данным, который возникает в циклах, как показано в следующем примере.

Рассмотрим цикл:

```
for (i=1; i<=100; i=i+1)  {
 A[i] = A[i] + B[i];    /* S1 */
 B[i+1] = C[i] + D[i];  /* S2 */
}
```

Оператор S1 использует значение, которое присваивается оператором S2 в предыдущей итерации, так что имеет место зависимость между S2 и S1 между итерациями.

Несмотря на эту зависимость, этот цикл может быть сделан параллельным. Как и в более раннем цикле эта зависимость не циклическая: ни один из операторов не зависит сам от себя и хотя S1 зависит от S2, S2 не зависит от S1. Цикл является параллельным, если только отсутствует циклическая зависимость.
Хотя в вышеприведенном цикле отсутствуют циклические зависимости, чтобы выявить параллелизм, его необходимо преобразовать в другую структуру. Здесь следует сделать два важных замечания:

1. 1. Зависимость от S1 к S2 отсутствует. Если бы она была, то в зависимостях появился бы цикл и цикл не был бы параллельным. Вследствие отсутствия других зависимостей, перестановка двух операторов не будет влиять на выполнение оператора S2.
2. 2. В первой итерации цикла оператор S1 зависит от значения B[1], вычисляемого перед началом цикла.

Эти два замечания позволяют нам заменить выше приведенный цикл следующей последовательностью:

```
A[1] = A[1] + B[1];
for (i=1; i<=99; i=i+1)  {
B[i+1] = C[i] + D[i];
A[i+1] = A[i+1] + B[i+1];
}
B[101] = C[100] + D[100];
```

Теперь итерации цикла могут выполняться с перекрытием, при условии, что операторы в каждой итерации выполняются в заданном порядке. Имеется множество такого рода преобразований, которые реструктурируют цикл для выявления параллелизма.
Основное внимание в оставшейся части этой главы сосредоточено на методах выявления параллелизма уровня команд. Зависимости по данным в откомпилированных программах представляют собой ограничения, которые оказывают влияние на то, какая степень параллелизма может быть использована. Вопрос заключается в том, чтобы подойти к этому пределу путем минимизации действительных конфликтов и связанных с ними приостановок конвейера. Методы, которые мы изучаем, становятся все более изощренными в стремлении использования всего доступного параллелизма при поддержании истинных зависимостей по данным в коде программы. Как компилятор, так и аппаратура здесь играют свою роль: компилятор старается устранить или минимизировать зависимости, в то время как аппаратура старается предотвратить превращение зависимостей в приостановки конвейера.

Основы планирования загрузки конвейера и разворачивание циклов

Для поддержания максимальной загрузки конвейера должен использоваться параллелизм уровня команд, основанный на выявлении последовательностей несвязанных команд, которые могут выполняться в конвейере с совмещением. Чтобы избежать приостановки конвейера зависимая команда должна быть отделена от исходной команды на расстояние в тактах, равное задержке конвейера для этой исходной команды. Способность компилятора выполнять подобное планирование зависит как от степени параллелизма уровня команд, доступного в программе, так и от задержки функциональных устройств в конвейере. В рамках этой главы мы будем предполагать задержки, показанные на рис. 6.2, если только явно не установлены другие задержки. Мы предполагаем, что условные переходы имеют задержку в один такт, так что команда, следующая за командой перехода, не может быть определена в течение одного такта после команды условного перехода. Мы предполагаем, что функциональные устройства полностью конвейеризованы или дублированы (столько раз, какова глубина конвейера), так что операция любого типа может выдаваться для выполнения в каждом такте, и структурные конфликты отсутствуют.

Команда, вырабатывающая результат	Команда, использующая результат	Задержка в тактах
Операция АЛУ с ПТ	Другая операция АЛУ с ПТ	3
Операция АЛУ с ПТ	Запись двойного слова	2
Загрузка двойного слова	Другая операция АЛУ с ПТ	1
Загрузка двойного слова	Запись двойного слова	0

Рис. 6.2.

В данном коротком разделе мы рассмотрим вопрос о том, каким образом компилятор может увеличить степень параллелизма уровня команд путем разворачивания циклов. Для иллюстрации этих методов мы будем использовать простой цикл, который добавляет скалярную величину к вектору в памяти; это параллельный цикл, поскольку зависимость между итерациями цикла отсутствует. Мы предполагаем, что первоначально в регистре R1 находится адрес последнего элемента вектора (например, элемент с наибольшим адресом), а в регистре F2 - скалярная величина, которая должна добавляться к каждому элементу вектора. Программа для машины, не рассчитанная на использование конвейера, будет выглядеть примерно так:

```
Loop:  LD    F0,0(R1)     ;F0=элемент вектора
       ADDD  F4,F0,F2     ;добавляет скаляр из F2
       SD    0(R1),F4     ;запись результата
       SUBI  R1,R1,#8     ;пересчитать указатель
                          ;8 байт (в двойном слове)
       BNEZ  R1, Loop     ;переход R1!=нулю
```

Для упрощения мы предполагаем, что массив начинается с ячейки 0. Если бы он находился в любом другом месте, цикл потребовал бы наличия одной дополнительной целочисленной команды для выполнения сравнения с регистром R1.

Рассмотрим работу этого цикла при выполнении на простом конвейере с задержками, показанными на рис. 6.2.

Если не делать никакого планирования, работа цикла будет выглядеть следующим образом:

```
                          Такт выдачи
Loop:  LD    F0,0(R1)          1
       Приостановка            2
       ADDD  F4,F0,F2          3
       Приостановка            4
       Приостановка            5
       SD    0(R1),F4          6
       SUBI  R1,R1,#8          7
       BNEZ  R1,Loop           8
       Приостановка            9
```

Для его выполнения потребуется 9 тактов на итерацию: одна приостановка для команды LD, две для команды ADDD, и одна для задержанного перехода. Мы можем спланировать цикл так, чтобы получить

```
Loop:  LD     F0,0(R1)                                  1
       Приостановка                                     2
       ADDD   F4,F0,F2                                  3
       SUBI   R1,R1,#8                                  4
       BNEZ   R1,Loop      ;задержанный переход         5
       SD     8(R1),F4     ;команда изменяется, когда   6
                           ;меняется местами с командой SUB1
```

Время выполнения уменьшилось с 9 до 6 тактов.

Заметим, что для планирования задержанного перехода компилятор должен определить, что он может поменять местами команды SUB1 и SD путем изменения адреса в команде записи SD: Адрес был равен 0(R1), а теперь равен 8(R1). Это не тривиальная задача, поскольку большинство компиляторов будут видеть, что команда SD зависит от SUB1, и откажутся от такой перестановки мест. Более изощренный компилятор смог бы рассчитать отношения и выполнить перестановку. Цепочка зависимостей от команды LD к команде ADDD и далее к команде SD определяет количество тактов, необходимое для данного цикла.

В вышеприведенном примере мы завершаем одну итерацию цикла и выполняем запись одного элемента вектора каждые 6 тактов, но действительная работа по обработке элемента вектора отнимает только 3 из этих 6 тактов (загрузка, сложение и запись). Оставшиеся 3 такта составляют накладные расходы на выполнение цикла (команды SUB1, BNEZ и приостановка). Чтобы устранить эти три такта нам нужно иметь больше операций в цикле относительно числа команд, связанных с накладными расходами. Одним из наиболее простых методов увеличения числа команд по отношению к команде условного перехода и команд, связанных с накладными расходами, является разворачивание цикла. Такое разворачивание выполняется путем многократной репликации (повторения) тела цикла и коррекции соответствующего кода конца цикла.

Разворачивание циклов может также использоваться для улучшения планирования. В этом случае, мы можем устранить приостановку, связанную с задержкой команды загрузки путем создания дополнительных независимых команд в теле цикла. Затем компилятор может планировать эти команды для помещения в слот задержки команды загрузки. Если при разворачивании цикла мы просто реплицируем команды, то результирующие зависимости по именам могут помешать нам эффективно спланировать цикл. Таким образом, для разных итераций хотелось бы использовать различные регистры, что увеличивает требуемое число регистров.

Представим теперь этот цикл развернутым так, что имеется четыре копии тела цикла, предполагая, что R1 первоначально кратен 4. Устраним при этом любые очевидные излишние вычисления и не будем пользоваться повторно никакими регистрами.

Ниже приведен результат, полученный путем слияния команд SUB1 и выбрасывания ненужных операций BNEZ, которые дублируются при разворачивании цикла.

```
Loop:  LD     F0,0(R1)
       ADDD   F4,F0,F2
       SD     0(R1),F4       ;выбрасывается SUB1 и BNEZ
       LD     F6,-8(R1)
       ADDD   F8,F6,F2
       SD     -8(R1),F8      ;выбрасывается SUB1 и BNEZ
       LD     F10,-16(R1)
       ADDD   F12,F10,F2
```

```
        SD      -16(R1),F12     ;выбрасывается SUB1 и BNEZ
        LD      F14,-24(R1)
        ADDD    F16,F14,F2
        SD      -24(R1),F16
        SUB1    R1,R1,#32
        BNEZ    R1, Loop
```

Мы ликвидировали три условных перехода и три операции декрементирования R1. Адреса команд загрузки и записи были скорректированы так, чтобы позволить слить команды SUB1 в одну команду по регистру R1. При отсутствии планирования за каждой командой здесь следует зависимая команда, и это будет приводить к приостановкам конвейера. Этот цикл будет выполняться за 27 тактов (на каждую команду LD потребуется 2 такта, на каждую команду ADDD - 3, на условный переход - 2 и на все другие команды 1 такт) или по 6.8 такта на каждый из четырех элементов. Хотя эта развернутая версия в такой редакции медленнее, чем оптимизированная версия исходного цикла, после оптимизации самого развернутого цикла ситуация изменится. Обычно разворачивание циклов выполняется на более ранних стадиях процесса компиляции, так что избыточные вычисления могут быть выявлены и устранены оптимизатором.

В реальных программах мы обычно не знаем верхней границы цикла. Предположим, что она равна n и мы хотели бы развернуть цикл так, чтобы иметь k копий тела цикла. Вместо единственного развернутого цикла мы генерируем пару циклов. Первый из них выполняется (n mod k) раз и имеет тело первоначального цикла. Развернутая версия цикла окружается внешним циклом, который выполняется (n div k) раз.

В вышеприведенном примере разворачивание цикла увеличивает производительность этого цикла путем устранения команд, связанных с накладными расходами цикла, хотя оно заметно увеличивает размер программного кода. Насколько увеличится производительность, если цикл будет оптимизироваться?

Ниже представлен развернутый цикл из предыдущего примера после оптимизации.

```
Loop:   LD      F0,0(R1)
        LD      F6,-8(R1)
        LD      F10,-16(R1)
        LD      F14,-24(R1)
        ADDD    F4,F0,F2
        ADDD    F8,F6,F2
        ADDD    F12,F10,F2
        ADDD    F16,F14,F2
        SD      0(R1),F4
        SD      -8(R1),F8
        SD      -16(R1),F12
        SUB1    R1,R1,#32
        BNEZ    R1, Loop
        SD      8(R1),F16       ; 8 - 32 = -24
```

Время выполнения развернутого цикла снизилось до 14 тактов или до 3.5 тактов на элемент, по сравнению с 6.8 тактов на элемент до оптимизации, и по сравнению с 6 тактами при оптимизации без разворачивания цикла.

Выигрыш от оптимизации развернутого цикла даже больше, чем от оптимизации первоначального цикла. Это произошло потому, что разворачивание цикла выявило больше вычислений, которые могут быть оптимизированы для минимизации приостановок конвейера; приведенный выше программный код выполняется без приостановок. При подобной оптимизации цикла необходимо осознавать, что команды загрузки и записи являются независимыми и могут чередоваться. Анализ зависимостей по данным позволяет нам определить, являются ли команды загрузки и записи независимыми.

Разворачивание циклов представляет собой простой, но полезный метод увеличения размера линейного кодового фрагмента, который может эффективно оптимизироваться. Это преобразование полезно на множестве машин от простых конвейеров, подобных рассмотренному ранее, до суперскалярных конвейеров, которые обеспечивают выдачу для выполнения более одной команды в такте. В следующем разделе рассмотрены методы, которые используются аппаратными средствами для динамического планирования загрузки конвейера и сокращения приостановок из-за конфликтов типа RAW, аналогичные рассмотренным выше методам компиляции.

5.2. **6.2. Устранение зависимостей по данным и механизмы динамического планирования**

Основная идея динамической оптимизации

Главным ограничением методов конвейерной обработки, которые мы рассматривали ранее, является выдача для выполнения команд строго в порядке, предписанном программой: если выполнение какой-либо команды в конвейере приостанавливалось, следующие за ней команды также приостанавливались. Таким образом, при наличии зависимости между двумя близко расположенными в конвейере командами возникала приостановка обработки многих команд. Но если имеется несколько функциональных устройств, многие из них могут оказаться незагруженными. Если команда j зависит от длинной команды i, выполняющейся в конвейере, то все команды, следующие за командой j должны приостановиться до тех пор, пока команда i не завершится и не начнет выполняться команда j. Например, рассмотрим следующую последовательность команд:

```
DIVD  F0,F2,F4
ADDD  F10,F0,F8
SUBD  F8,F8,F14
```

Команда SUBD не может выполняться из-за того, что зависимость между командами DIVD и ADDD привела к приостановке конвейера. Однако команда SUBD не имеет никаких зависимостей от команд в конвейере. Это ограничение производительности, которое может быть устранено снятием требования о выполнении команд в строгом порядке.

В рассмотренном нами конвейере структурные конфликты и конфликты по данным проверялись во время стадии декодирования команды (ID). Если команда могла нормально выполняться, она выдавалась с этой ступени конвейера в следующие. Чтобы позволить начать выполнение команды SUBD из предыдущего примера, необходимо разделить процесс выдачи на две части: проверку наличия структурных конфликтов и ожидание отсутствия конфликта по данным. Когда мы выдаем команду для выполнения, мы можем осуществлять проверку наличия структурных конфликтов; таким образом, мы все еще используем упорядоченную выдачу команд. Однако мы хотим начать выполнение команды, как только станут доступными ее операнды. Таким образом, конвейер будет осуществлять неупорядоченное выполнение команд, которое означает и неупорядоченное завершение команд.

Неупорядоченное завершение команд создает основные трудности при обработке исключительных ситуаций. В рассматриваемых в данном разделе машинах с динамическим планированием потока команд прерывания будут неточными, поскольку команды могут завершиться до того, как выполнение более ранней выданной команды вызовет исключительную ситуацию. Таким образом, очень трудно повторить запуск после прерывания. Вместо того, чтобы рассматривать эти проблемы в данном разделе, мы обсудим возможные решения для реализации точных прерываний позже в контексте машин, использующих планирование по предположению.

Чтобы реализовать неупорядоченное выполнение команд, мы расщепляем ступень ID на две ступени:

1. Выдача - декодирование команд, проверка структурных конфликтов.
2. Чтение операндов - ожидание отсутствия конфликтов по данным и последующее чтение операндов.

Затем, как и в рассмотренном нами конвейере, следует ступень EX. Поскольку выполнение команд ПТ может потребовать нескольких тактов в зависимости от типа операции, мы должны знать, когда команда начинает выполняться и когда заканчивается. Это позволяет нескольким командам выполняться в один и тот же момент времени. В дополнение к этим изменениям структуры конвейера мы изменим и структуру функциональных устройств, варьируя количество устройств, задержку операций и степень конвейеризации функциональных устройств так, чтобы лучше использовать эти методы конвейеризации.

Динамическая оптимизация с централизованной схемой обнаружения конфликтов

В конвейере с динамическим планированием выполнения команд все команды проходят через ступень выдачи строго в порядке, предписанном программой (упорядоченная выдача). Однако они могут приостанавливаться и обходить друг друга на второй ступени (ступени чтения операндов) и тем самым поступать на ступени выполнения неупорядочено. Централизованная схема обнаружения конфликтов представляет собой метод, допускающий неупорядоченное выполнение команд при наличии достаточных ресурсов и отсутствии зависимостей по данным. Впервые подобная схема была применена в компьютере CDC 6600.

Прежде чем начать обсуждение возможности применения подобных схем, важно заметить, что конфликты типа WAR, отсутствующие в простых конвейерах, могут появиться при неупорядоченном выполнении команд. В ранее приведенном примере регистром результата для команды SUBD является регистр R8, который одновременно является источником операнда для команды ADDD. Поэтому здесь между командами ADDD и SUBD имеет место антизависимость: если конвейер выполнит команду SUBD раньше команды ADDD, он нарушит эту антизависимость. Этот конфликт WAR можно обойти, если выполнить два правила: (1) читать регистры только во время стадии чтения операндов и (2) поставить в очередь операцию ADDD вместе с копией ее операндов. Чтобы избежать нарушений зависимостей по выходу конфликты типа WAW (например, это могло произойти, если бы регистром результата команды SUBD была бы регистр F10) все еще должны обнаруживаться. Конфликты типа WAW могут быть устранены с помощью приостановки выдачи команды, регистр результата которой совпадает с уже используемым в конвейере.

Задачей централизованной схемы обнаружения конфликтов является поддержание выполнения команд со скоростью одна команда за такт (при отсутствии структурных конфликтов) посредством как можно более раннего начала выполнения команд. Таким образом, когда команда в начале очереди приостанавливается, другие команды могут выдаваться и выполняться, если они не зависят от уже выполняющейся или приостановленной команды. Централизованная схема несет полную ответственность за выдачу и выполнение команд, включая обнаружение конфликтов. Подобное неупорядоченное выполнение команд требует одновременного нахождения нескольких команд на стадии выполнения. Этого можно достигнуть двумя способами: реализацией в процессоре либо множества неконвейерных функциональных устройств, либо путем конвейеризации всех функциональных устройств. Обе эти возможности, по сути, эквивалентны с точки зрения организации управления. Поэтому предположим, что в машине имеется несколько неконвейерных функциональных устройств.

Машина CDC 6600 имела 16 отдельных функциональных устройств (4 устройства для операций с плавающей точкой, 5 устройств для организации обращений к основной памяти и 7 устройств для целочисленных операций). В нашем случае централизованная схема обнаружения конфликтов имеет смысл только для устройства плавающей точки. Предположим, что имеются два умножителя, один сложитель, одно устройство деления и одно целочисленное устройство для всех операций обращения к памяти, переходов и целочисленных операций. Хотя устройств в этом примере гораздо меньше, чем в CDC 6600, он достаточно мощный для демонстрации основных принципов работы. Поскольку как наша машина, так и CDC 6600 являются машинами с операциями регистр-регистр (операциями загрузки/записи), в обеих машинах методика практически одинаковая. На рис. 6.3 показана подобная машина.

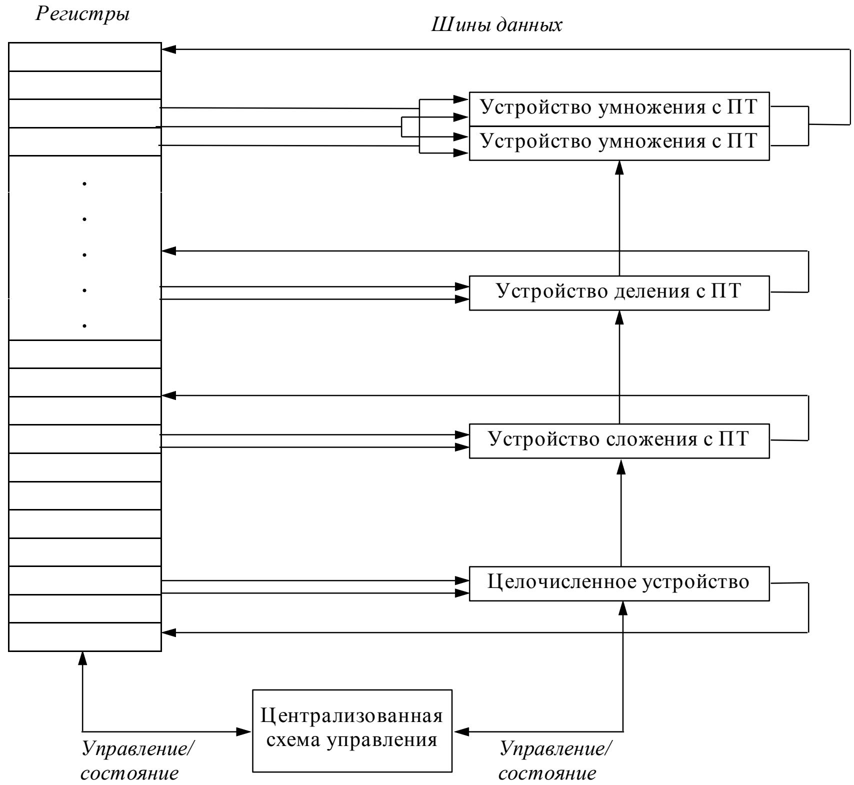


Рис. 6.3. Централизованная схема управления

Каждая команда проходит через централизованную схему обнаружения конфликтов, которая определяет зависимости по данным; этот шаг соответствует стадии выдачи команд и заменяет часть стадии ID в нашем конвейере. Эти зависимости определяют затем моменты времени, когда команда может читать свои операнды и начинать выполнение операции. Если централизованная схема решает, что команда не может немедленно выполняться, она следит за всеми изменениями в аппаратуре и решает, когда команда сможет выполняться. Эта же централизованная схема определяет также, когда команда может записать результат в свой регистр результата. Таким образом, все схемы обнаружения и разрешения конфликтов здесь выполняются устройством центрального управления.

Каждая команда проходит четыре стадии своего выполнения. (Поскольку в данный момент мы интересуемся операциями плавающей точки, мы не рассматриваем стадию обращения к памяти). Рассмотрим эти стадии сначала неформально, а затем детально рассмотрим, как централизованная схема поддерживает необходимую информацию, которая определяет обработку при переходе с одной стадии на другую. Следующие четыре стадии заменяют стадии ID, EX и WB в стандартном конвейере:

1. *Выдача.* Если функциональное устройство, необходимое для выполнения команды, свободно и никакая другая выполняющаяся команда не использует тот же самый регистр результата, централизованная схема выдает команду в функциональное устройство и обновляет свою внутреннюю структуру данных. Поскольку никакое другое работающее функциональное устройство не может записать результат в регистр результата нашей команды, мы гарантируем, что конфликты типа WAW не могут появляться. Если существует структурный конфликт или конфликт типа WAW, выдача команды блокируется, и никакие следующие команды не будут выдаваться на выполнение до тех пор, пока эти конфликты существуют. Эта стадия заменяет часть стадии ID в нашем конвейере.
2. *Чтение операндов.* Централизованная схема следит за возможностью выборки источников операндов для соответствующей команды. Операнд-источник доступен, если отсутствует выполняющаяся команда, которая записывает результат в этот регистр или если в данный момент времени в регистр, содержащий операнд, выполняется запись из работающего функционального устройства. Если операнды-источники доступны, централизованная схема сообщает функциональному устройству о необходимости чтения операндов из регистров и начале выполнения операции. Централизованная схема разрешает конфликты RAW на этой стадии динамически, и команды могут посылаться для выполнения не в порядке, предписанном программой. Эта стадия, совместно со стадией выдачи, завершает работу стадии ID простого конвейера.
3. *Выполнение.* Функциональное устройство начинает выполнение операции после получения операндов. Когда результат готов оно уведомляет централизованную схему управления о том, что оно завершило выполнение операции. Эта стадия заменяет стадию EX и занимает несколько тактов в рассмотренном ранее конвейере.
4. *Запись результата.* Когда централизованная схема управления узнает о том, что функциональное устройство завершило выполнение операции, она проверяет существование конфликта типа WAR. Конфликт типа WAR существует, если имеется последовательность команд, аналогичная представленной в нашем примере с командами ADDF и SUBF. В том примере мы имели следующую последовательность команд:

```
DIVF  F0,F2,F4
ADDF  F10,F0,F8
SUBF  F8,F8,F14
```

Команда ADDF имеет операнд-источник F8, который является тем же самым регистром, что и регистр результата команды SUBF. Но в действительности команда ADDF зависит от предыдущей команды. Централизованная схема управления будет блокировать выдачу команды SUBF до тех пор, пока команда ADDF не прочитает свои операнды. Тогда в общем случае завершающейся команде не разрешается записывать свои результаты если:

- имеется команда, которая не прочитала свои операнды,
- один из операндов является регистром результата завершающейся команды.

Если этот конфликт типа WAR не существует, централизованная схема управления сообщает функциональному устройству о необходимости записи результата в регистр назначения. Эта стадия заменяет стадию WB в простом конвейере.

Основываясь на своей собственной структуре данных, централизованная схема управления управляет продвижением команды с одной ступени на другую, взаимодействуя с функциональными устройствами. Но имеется небольшое усложнение: в регистровом файле имеется только ограниченное число магистралей для

операндов-источников и магистралей для записи результата. Централизованная схема управления должна гарантировать, что количество функциональных устройств, которым разрешено продолжать работу на ступенях 2 и 4 не превышает числа доступных шин. Мы не будем вдаваться в дальнейшие подробности и упомянем лишь, что CDC 6600 решала эту проблему путем объединения 16 функциональных устройств друг с другом в четыре группы и поддержки для каждой группы устройств набора шин, называемых магистралями данных (data trunks). Только одно устройство в группе могло читать операнды или записывать свой результат в течение одного такта.
Общая структура регистров состояния устройства централизованного управления показана на рисунке 6.4. Она состоит из 3-х частей:

1. 1. Состояние команды - показывает каждый из четырех этапов выполнения команды.
2. 2. Состояние функциональных устройств - имеются 9 полей, описывающих состояние каждого функционального устройства:

   Занятость - показывает, занято устройство или свободно

   Op - выполняемая в устройстве операция

   F_i - регистр результата

   F_j, F_k - регистры-источники операндов

   Q_j, Q_k - функциональные устройства, вырабатывающие результат для записи в регистры F_j, F_k

   R_j, R_k - признаки готовности операндов в регистрах F_j, F_k
3. 3. Состояние регистров результата - показывает функциональное устройство, которое будет записывать в каждый из регистров. Это поле устанавливается в ноль, если отсутствуют команды, записывающие результат в данный регистр.

Интересным вопросом является стоимость и преимущества централизованного управления. Разработчики CDC 6600 оценивают улучшение производительности для программ на Фортране в 1.7 раза, а для вручную запрограммированных на языке ассемблера программ в 2.5 раза. Однако эти оценки делались в то время, когда отсутствовали программные средства планирования загрузки конвейера, полупроводниковая основная память и кэш-память (с малым временем доступа). Централизованная схема управления CDC 6600 имела примерно столько же логических схем, что и одно из функциональных устройств, что на удивление мало. Основная стоимость определялась большим количеством шин (магистралей) - примерно в четыре раза больше по сравнению с машиной, которая выполняла бы команды в строгом порядке, заданном программой.
Централизованная схема управления не обрабатывает несколько ситуаций. Например, когда команда записывает свой результат, зависимая команда в конвейере должна дожидаться разрешения обращения к регистровому файлу, поскольку все результаты всегда записываются в регистровый файл, и никогда не используется методика "ускоренной пересылки". Это увеличивает задержку и ограничивает возможность инициирования нескольких команд, ожидающих результата. Что касается CDC 6600, то конфликты типа WAW являются очень редкими, так что приостановки, которые они вызывают, возможно, не являются существенными. Однако в следующем разделе мы увидим, что динамическое планирование дает возможность совмещенного выполнения нескольких итераций цикла. Чтобы это делать эффективно, требуется схема обработки конфликтов типа WAW, которые вероятно увеличиваются по частоте при совмещении выполнения нескольких итераций.

Состояние команд				
Команда	**Выдача**	**Чтение операндов**	**Завершение выполнения**	**Запись результата**
`LD F6,34(R2)`	√	√	√	√
`LD F2,45(R3)`	√	√	√	
`MULTD F0,F2,F4`	√			
`SUBD F8,F6,F2`	√			
`DIVD F10,F0,F6`	√			
`ADD F6,F8,F2`				

Состояние функциональных устройств									
Имя	**Занятость**	**Op**	**F_i**	**F_j**	**F_k**	**Q_j**	**Q_k**	**R_j**	**R_k**
Integer	Да	Load	F2	R3					
Mult1	Да	Mult	F0	F2	F4			Нет	Да
Mult2	Нет								
Add	Да	Sub	F8	F6	F2		Integer	Да	Нет
Divide	Да	Div	F10	F0	F6	Mult1		Нет	Да

Состояние регистра результата									
	F0	**F2**	**F4**	**F6**	**F8**	**F10**	**F12**	**. . .**	**F30**
FU	Mult1	Integer			Add	Divide			

Рис. 6.4. Регистры состояния централизованной схемы управления

Другой подход к динамическому планированию - алгоритм Томасуло

Другой подход к параллельному выполнению команд при наличии конфликтов был использован в устройстве плавающей точки в машине IBM 360/91. Эта схема приписывается Р. Томасуло и названа его именем. Разработка IBM 360/91 была завершена спустя три года после выпуска CDC 6600, прежде чем кэш-память появилась в коммерческих машинах. Задачей IBM было достижение высокой производительности на операциях с плавающей точкой, используя набор команд и компиляторы, разработанные для всего семейства 360, а не только для приложений с интенсивным использованием плавающей точки. Архитектура 360 предусматривала только четыре регистра плавающей точки двойной точности, что ограничивало эффективность планирования кода компилятором. Этот факт был другой мотивацией подхода Томасуло. Наконец, машина IBM 360/91 имела большое время обращения к памяти и большие задержки выполнения операций плавающей точки, преодолеть которые и был призван разработанный Томасуло алгоритм. В конце раздела мы увидим, что алгоритм Томасуло может также поддерживать совмещенное выполнение нескольких итераций цикла.

Мы поясним этот алгоритма на примере устройства ПТ. Основное различие между нашим конвейером ПТ и конвейером машины IBM/360 заключается в наличии в последней машине команд типа регистр-память. Поскольку алгоритм Томасуло использует функциональное устройство загрузки, не требуется значительных изменений, чтобы добавить режимы адресации регистр-память; основное добавление - другая шина. IBM 360/91 имела также конвейерные функциональные устройства, а не несколько функциональных устройств. Единственное отличие между ними заключается в том, что конвейерное функциональное устройство может начинать выполнение только одной операции в каждом такте. Поскольку реально отсутствуют фундаментальные отличия, мы описываем алгоритм, как если бы имели место несколько функциональных устройств. IBM 360/91 могла выполнять одновременно три операции сложения ПТ и две операции умножения ПТ. Кроме того, в процессе выполнения могли находиться до 6 операций загрузки ПТ, или обращений к памяти, и до трех операций записи ПТ. Для реализации этих функций использовались буфера данных загрузки и буфера данных записи. Хотя мы не будем обсуждать устройства загрузки и записи, необходимо добавить буфера для операндов.

Схема Томасуло имеет много общего со схемой централизованного управления CDC 6600, однако, имеются и существенные отличия. Во-первых, обнаружение конфликтов и управление выполнением являются распределенными - станции резервирования (reservation stations) в каждом функциональном устройстве определяют, когда команда может начать выполняться в данном функциональном устройстве. В CDC 6600 эта функция централизована. Во-вторых, результаты операций посылаются прямо в функциональные устройства, а не проходят через регистры. В IBM 360/91 имеется общая шина результатов операций (которая называется общей шиной данных (common data bus - CDB)), которая позволяет производить одновременную загрузку всех устройств, ожидающих операнда. CDC 6600 записывает результаты в регистры, за которые ожидающие функциональные устройства могут соперничать. Кроме того, CDC 6600 имеет несколько шин завершения операций (две в устройстве ПТ), а IBM 360/91 - только одну.

На рис. 6.5 представлена основная структура устройства ПТ на базе алгоритма Томасуло. Никаких таблиц управления выполнением не показано. Станции резервирования хранят команды, которые выданы и ожидают выполнения в соответствующем функциональном устройстве, а также информацию, требующуюся для управления командой, когда ее выполнение началось в функциональном устройстве. Буфера загрузки и записи хранят данные, поступающие из памяти и записываемые в память. Регистры ПТ соединены с функциональными устройствами парой шин и одной шиной с буферами записи. Все результаты из функциональных устройств и из памяти посылаются на общую шину данных, которая связана с входами всех устройств за исключением буфера загрузки. Все буфера и станции резервирования имеют поля тегов, используемых для управления конфликтами.

Прежде чем описывать детали станций резервирования и алгоритм, рассмотрим все стадии выполнения команды. В отличие от централизованной схемы управления, имеется всего три стадии:

1. 1. Выдача - Берет команду из очереди команд ПТ. Если операция является операцией ПТ, выдает ее при наличии свободной станции резервирования и посылает операнды на станцию резервирования, если они находятся в регистрах. Если операция является операцией загрузки или записи, она может выдаваться при наличии свободного буфера. При отсутствии свободной станции резервирования или свободного буфера возникает структурный конфликт, и команда приостанавливается до тех пор, пока не освободится станция резервирования или буфер.
2. 2. Выполнение - Если один или более операндов команды не доступны по каким-либо причинам, контролируется состояние CDB и ожидается завершение вычисления значений нужного регистра. На этой стадии выполняется контроль конфликтов типа RAW. Когда оба операнда доступны, выполняется операция.
3. 3. Запись результата - Когда становится доступным результат, он записывается на CDB и оттуда в регистры и любое функциональное устройство, ожидающее этот результат.

Хотя эти шаги в основном похожи на аналогичные шаги в централизованной схеме управления, имеются три важных отличия. Во-первых, отсутствует контроль конфликтов типа WAW и WAR - они устраняются как побочный эффект алгоритма. Во-вторых, для трансляции результатов используется CDB, а не схема ожидания готовности регистров. В-третьих, устройства загрузки и записи рассматриваются как основные функциональные устройства.

Структуры данных, используемые для обнаружения и устранения конфликтов, связаны со станциями резервирования, регистровым файлом и буферами загрузки и записи. Хотя с разными объектами связана разная информация, все устройства, за исключением буферов загрузки, содержат в каждой строке поле тега. Это поле тега представляет собой четырехбитовое значение, которое обозначает одну из пяти станций резервирования или один из шести буферов загрузки. Поле тега используется для описания того, какое функциональное устройства будет поставлять результат, нужный в качестве источника операнда. Неиспользуемые значения, такие как ноль, показывают, что операнд уже доступен. Важно помнить, что теги в схеме Томасуло ссылаются на буфера или устройства, которые будут поставлять результат. Когда команда выдается в станцию резервирования, номера регистров исключаются из рассмотрения.

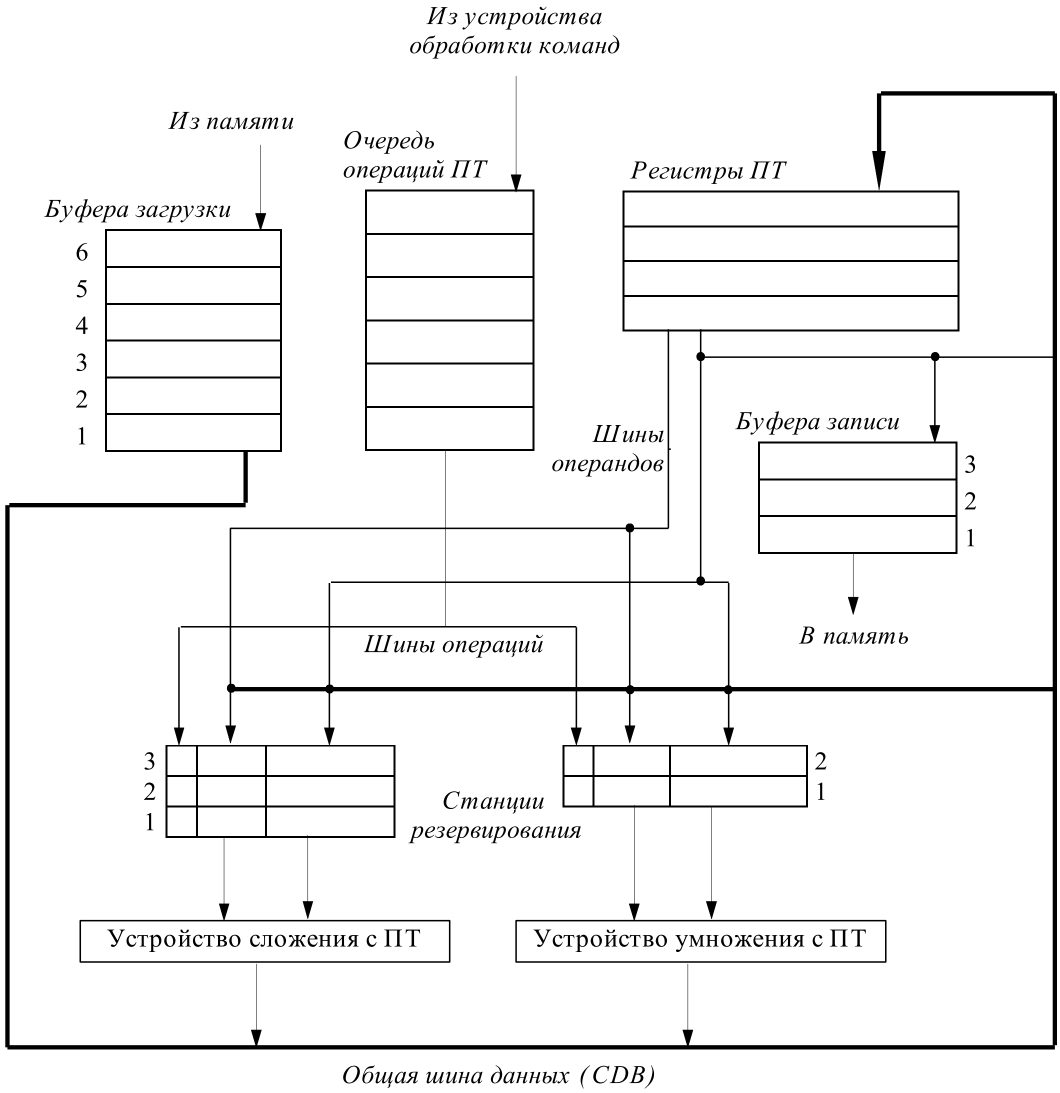


Рис. 6.5. Структура устройства ПТ на основе алгоритма Томасуло

Каждая станция резервирования содержит шесть полей:

- • O_p - Операция, которая должна выполняться над источниками операндов S_1 и S_2;
- • Q_j,Q_k - станции резервирования, которые будут вырабатывать соответствующий операнд-источник; нулевое значение показывает, что операнд-источник уже доступен в V_j или V_k, или не является обязательным. IBM 360/91 называет их SINKunit и SOURCEunit.
- • V_j,V_k - значение операндов-источников. Они называются SINK и SOURCE в IBM 360/91. Заметим, что для каждого операнда являются действительными только одно из полей либо поле V, либо поле Q.
- • Занято - Показывает, что данная станция резервирования и ее соответствующее функциональное устройство заняты.

Регистровый файл и буфер записи имеют поле Q_i:

- • Q_i - номер функционального устройства, которое будет вырабатывать значение, которое надо записать в регистр или память. Если значение Q_i равно нулю, то это означает, что ни одна текущая активная команда не вычисляет результат для данного регистра или буфера. Для регистра это означает, что значение определяется содержимым регистра.

В каждом из буферов загрузки и записи требуется поле занятости, показывающее, когда соответствующий буфер становится доступным в результате завершения загрузки или записи, назначенных на этот буфер. Буфер записи имеет также поле V для хранения значения, которое должно быть записано в память.
Прежде, чем мы исследуем алгоритм в деталях, давайте рассмотрим, как выглядят системные таблицы для следующей последовательности команд:

```
1. LF       F6,34(R2)
2. LF       F2,45(R3)
3. MULTD    F0,F2,F4
4. SUBD     F8,F6,F2
5. DIVD     F10,F0,F6
6. ADDD     F6,F8,F2
```

Рис. 6.6 описывает станции резервирования, буфера загрузки и записи и регистровые теги. К именам add, mult и load добавлены номера, стоящие за тегами для этой станции резервирования - Add1 является тегом для результата из первого устройства сложения. Состояние каждой операции, которая выдана для выполнения, хранится в станции резервирования.

Имеются два важных отличия от централизованной схемы управления, которые заметны в этих таблицах. Во-первых, значение операнда хранится в станции резервирования в одном из полей V как только оно становится доступным; оно не считывается из регистрового файла во время выдачи команды. Во-вторых, команда ADDD выдана для выполнения. Ее выдача была заблокирована в централизованной схеме управления из-за наличия структурного конфликта.
Большие преимущества схемы Томасуло заключаются в (1) распределении логики обнаружения конфликтов, и (2) устранение приостановок, связанных с конфликтами типа WAW и WAR. Первое преимущество возникает из-за наличия распределенных станций резервирования и использования CDB. Если несколько команд ожидают один и тот же результат и каждая команда уже имеет свой другой операнд, то команды могут выдаваться одновременно посредством трансляции по CDB. В

централизованной схеме управления ожидающие команды должны читать свои операнды из регистров, когда станут доступными регистровые шины.

Состояние команд				
Команда		**Выдача**	**Выполнение**	**Запись результата**
LD	F6,34(R2)	√	√	√+
LD	F2,45(R3)	√	√	
MULTD	F0,F2,F4	√		
SUBD	F8,F6,F2	√		
DIVD	F10,F0,F6	√		
ADDD	F6,F8,F2	√		

Станции резервирования						
Имя	**Занятость**	**Op**	**V_j**	**V_k**	**Q_j**	**Q_k**
Add1	Да	SUB	Mem[34+Regs[R2]]			Load2
Add2	Да	ADD			Add1	Load2
Add3	Нет					
Mult1	Да	MULT		Regs[F4]	Load2	
Mult2	Да	DIV		Mem[34+Regs[R2]]	Mult1	

Состояние регистров									
Поле	**F0**	**F2**	**F4**	**F6**	**F8**	**F10**	**F12**	**. . .**	**F30**
Qi	Mult1	Load2		Add2	Add1	Mult2			

Рис. 6.6. Теги станций резервирования и регистров

Конфликты типа WAW и WAR устраняются путем переименования регистров, используя станции резервирования. Например, в нашей кодовой последовательности на рис. 6.6 мы выдали на выполнение как команду DIVD, так и команду ADDD, даже хотя имелся конфликт типа WAR по регистру F6. Конфликт устраняется одним из двух способов. Если команда, поставляющая значение для команды DIVD, завершилась, тогда V_k будет хранить результат, позволяя DIVD выполняться независимо от команды ADDD. С другой стороны, если выполнение команды LF не завершилось, то Q_k будет указывать на LOAD1 и команда DIVD будет независимой от ADDD. Таким образом, в любом случае команда ADDD может быть выдана, и начать выполняться. Любое использование результата команды MULTD будет указывать на станцию резервирования, позволяя ADDD завершить и записать свое значение в регистры без воздействия DIVD. Вскоре мы увидим пример устранения конфликта типа WAW.

Чтобы понять полную мощность устранения конфликтов типа WAW и WAR посредством динамического переименования регистров мы должны рассмотреть цикл. Рассмотрим следующую простую последовательность команд для умножения элементов вектора на скалярную величину, находящуюся в регистре F2:

```
Loop:   LD          F0,0(R1)
```

```
MULTD    F4,F0,F2
SD       0(R1),F4
SUBI     R1,R1,#8
BNEZ     R1,Loop      ; условный переход при R1 /=0
```

Со стратегией выполняемого перехода использование станций резервирования позволит сразу же продолжить выполнение нескольких итераций этого цикла. Это преимущество дается без статического разворачивания цикла программными средствами: в действительности цикл разворачивается динамически аппаратурой. В архитектуре 360 наличие всего 4 регистров ПТ сильно ограничивало бы использование статического разворачивания цикла. (Вскоре мы увидим, что при статическом разворачивании и планировании выполнения цикла для обхода взаимных блокировок требуется значительно большее число регистров). Алгоритм Томасуло поддерживает выполнение с перекрытием нескольких копий одного и того же цикла при наличии лишь небольшого числа регистров, используемых программой.

Давайте предположим, что мы выдали для выполнения все команды двух последовательных итераций цикла, но еще не завершилось выполнение ни одной операции загрузки/записи в память. Станции резервирования, таблицы состояния регистров и буфера загрузки/записи в этой точке показаны на рис. 6.7. (Здесь операции целочисленного АЛУ игнорируются, и предполагается, что условный переход был спрогнозирован как выполняемый). Когда система достигла такого состояния могут поддерживаться две копии цикла с CPI, близким к единице, при условии, что операции умножения могут завершиться за четыре такта. Если мы игнорируем накладные расходы цикла, которые не снижены в этой схеме, то полученный уровень производительности будет соответствовать тому, который мы могли бы достигнуть посредством статического разворачивания и планирования цикла компилятором при условии наличия достаточного числа регистров.

В этом примере показан дополнительный элемент, который является важным для того, чтобы алгоритм Томасуло работал. Команда загрузки из второй итерации цикла может легко закончиться раньше команды записи из первой итерации, хотя нормальный последовательный порядок отличается. Загрузка и запись могут надежно (безопасно) выполняться в различном порядке при условии, что загрузка и запись обращаются к разным адресам. Это контролируется путем проверки адресов в буфере записи каждый раз при выдаче команды загрузки. Если адрес команды загрузки соответствует одному из адресов в буфере записи мы должны остановиться и подождать до тех пор, пока буфер записи не получит требуемое значение; затем мы можем к нему обращаться или выбирать значение из памяти. Это динамическое сравнение адресов является альтернативой технике, которая может использоваться компилятором при перестановке команд загрузки и записи.

Эта динамическая схема может достигать очень высокой производительности при условии того, что стоимость переходов может поддерживаться небольшой. Этот вопрос мы будем рассматривать в следующем разделе. Главный недостаток этого подхода заключается в сложности схемы Томасуло, которая требует для своей реализации очень большого объема аппаратуры. Особенно это касается большого числа устройств ассоциативной памяти, которая должна работать с высокой скоростью, а также сложной логики управления. Наконец, увеличение производительности ограничивается наличием одной шины завершения (CDB). Хотя дополнительные шины CDB могут быть добавлены, каждая CDB должна взаимодействовать со всей аппаратурой конвейера, включая станции резервирования. В частности, аппаратуру ассоциативного сравнения необходимо дублировать на каждой станции для каждой CDB.

В схеме Томасуло комбинируются две различных методики: методика переименования регистров буферизация операндов-источников из регистрового файла. Буферизация источников операндов разрешает конфликты типа WAR, которые возникают когда операнды доступны в регистрах. Как мы увидим позже, возможно также устранять конфликты типа WAR посредством переименования регистра вместе с буферизацией результата до тех пор, пока остаются обращения к старой версии регистра; этот подход будет использоваться, когда мы будем обсуждать аппаратное выполнение по предположению.

Состояние команд				
Команда	**Номер итерации**	**Выдача**	**Выполнение**	**Запись результата**
`LD F0,0(R1)`	1	√	√	
`MULTD F4,F0,F2`	1	√		
`SD 0(R1),F4`	1	√		
`LD F0,0(R1)`	2	√	√	
`MULTD F4,F0,F2`	2	√		
`SD 0(R1),F4`	2	√		

Станции резервирования						
Имя	**Занятость**	**Fm**	**V_j**	**V_k**	**Q_j**	**Q_k**
Add1	Нет		Mem[34+Regs[R2]]			
Add2	Нет					
Add3	Нет					
Mult1	Да	MULT		Regs[F2]	Load1	
Mult2	Да	MULT		Regs[F2]	Load2	

Состояние регистров									
Поле	**F0**	**F2**	**F4**	**F6**	**F8**	**F10**	**F12**	**. . .**	**F30**
Qi	Load2		Mult2						

Буфера загрузки			
Поле	**Load1**	**Load2**	**Load3**
Адрес	Regs[R1]	Regs[R1]-8	
Заня-тость	Да	Да	Нет

Буфера записи			
Поле	**Store1**	**Store2**	**Store3**
Q_i	Mult1	Mult2	
Заня-тость	Да	Да	Нет
Адрес	Regs[R1]	Regs[R1]-8	

Рис. 6.7. Состояние станций резервирования, регистров и буферов загрузки/записи

Схема Томасуло является привлекательной, если разработчик вынужден делать конвейерную архитектуру, для которой трудно выполнить планирование кода или реализовать большое хранилище регистров. С другой стороны, преимущество подхода Томасуло возможно ощущается меньше, чем увеличение стоимости реализации, по сравнению с методами планирования загрузки конвейера средствами компилятора в машинах, ориентированных на выдачу для выполнения только одной команды в такте. Однако по мере того, как машины становятся все более агрессивными в своих возможностях выдачи команд, и разработчики сталкиваются с вопросами производительности кода, который трудно планировать (большинство кодов для нечисловых расчетов), методика типа переименования регистров и динамического планирования будет становиться все более важной. Позже в этой главе мы увидим, что эти методы являются одним из важных компонентов большинства схем для реализации аппаратного выполнения по предположению.

Ключевыми компонентами увеличения параллелизма уровня команд в алгоритме Томасуло являются динамическое планирование, переименование регистров и динамическое устранение неоднозначности обращений к памяти. Трудно оценить значение каждого из этих свойств по отдельности.

Динамической аппаратной технике планирования загрузки конвейера при наличии зависимостей по данным соответствует и динамическая техника для эффективной обработки переходов. Эта техника используется для двух целей: для прогнозирования того, будет ли переход выполняемым, и для возможно более раннего нахождения целевой команды перехода. Эта техника называется аппаратным прогнозированием переходов.

5.3. 6.3. Аппаратное прогнозирование направления переходов и снижение потерь на организацию переходов

Буфера прогнозирования условных переходов

Простейшей схемой динамического прогнозирования направления условных переходов является буфер прогнозирования условных переходов (branch-prediction buffer) или таблица "истории" условных переходов (branch history table). Буфер прогнозирования условных переходов представляет собой небольшую память, адресуемую с помощью младших разрядов адреса команды перехода. Каждая ячейка этой памяти содержит один бит, который говорит о том, был ли предыдущий переход выполняемым или нет. Это простейший вид такого рода буфера. В нем отсутствуют теги, и он оказывается полезным только для сокращения задержки перехода в случае, если эта задержка больше, чем время, необходимое для вычисления значения целевого адреса перехода. В действительности мы не знаем, является ли прогноз корректным (этот бит в соответствующую ячейку буфера могла установить совсем другая команда перехода, которая имела то же самое значение младших разрядов адреса). Но это не имеет значения. Прогноз - это только предположение, которое рассматривается как корректное, и выборка команд начинается по прогнозируемому направлению. Если же предположение окажется неверным, бит прогноза инвертируется. Конечно, такой буфер можно рассматривать как кэш-память, каждое обращение к которой является попаданием, и производительность буфера зависит от того, насколько часто прогноз применялся и насколько он оказался точным.

Однако простая однобитовая схема прогноза имеет недостаточную производительность. Рассмотрим, например, команду условного перехода в цикле, которая являлась выполняемым переходом последовательно девять раз подряд, а затем однажды невыполняемым. Направление перехода будет неправильно предсказываться при первой и при последней итерации цикла. Неправильный прогноз последней итерации цикла неизбежен, поскольку бит прогноза будет говорить, что переход "выполняемый" (переход был девять раз подряд выполняемым). Неправильный прогноз на первой итерации происходит из-за того, что бит прогноза инвертируется при предыдущем выполнении последней итерации цикла, поскольку в этой итерации переход был невыполняемым. Таким образом, точность прогноза для перехода, который выполнялся в 90% случаев, составила только 80% (2 некорректных прогноза и

8 корректных). В общем случае, для команд условного перехода, используемых для организации циклов, переход является выполняемым много раз подряд, а затем один раз оказывается невыполняемым. Поэтому однобитовая схема прогнозирования будет неправильно предсказывать направление перехода дважды (при первой и при последней итерации).

Для исправления этого положения часто используется схема двухбитового прогноза. В двухбитовой схеме прогноз должен быть сделан неверно дважды, прежде чем он изменится на противоположное значение. На рис. 6.8 представлена диаграмма состояний двухбитовой схемы прогнозирования направления перехода.

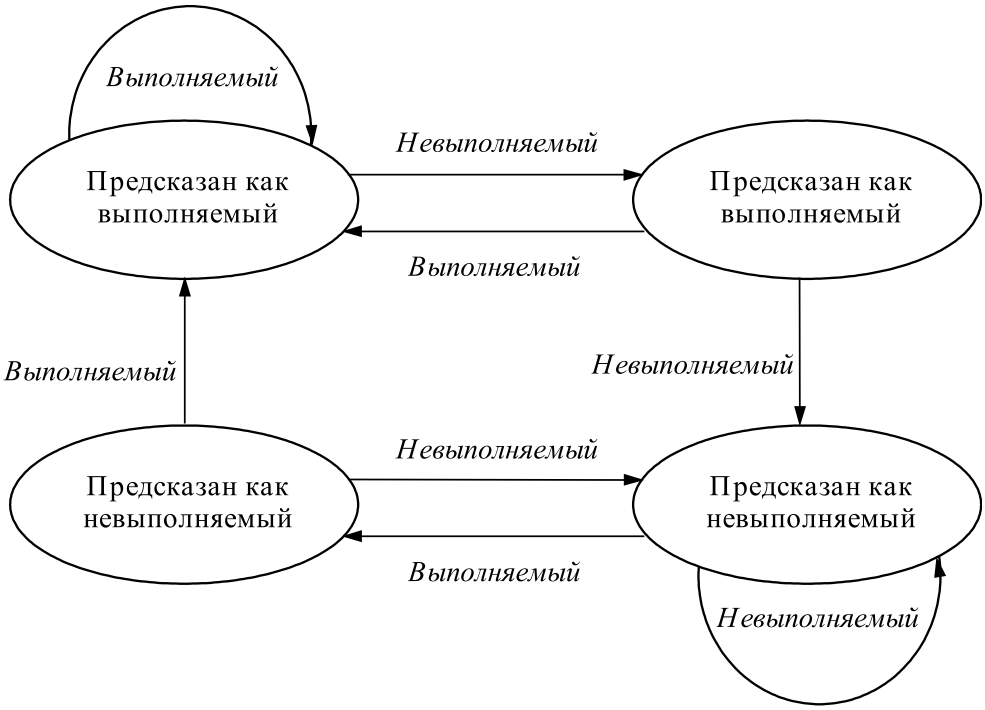


Рис. 6.8. Диаграмма состояния двухбитовой схемы прогнозирования

Двухбитовая схема прогнозирования в действительности является частным случаем более общей схемы, которая в каждой строке буфера прогнозирования имеет n-битовый счетчик. Этот счетчик может принимать значения от 0 до 2^n - 1. Тогда схема прогноза будет следующей:

- • Если значение счетчика больше или равно 2^{n-1} (точка на середине интервала), то переход прогнозируется как выполняемый. Если направление перехода предсказано правильно, к значению счетчика добавляется единица (если только оно не достигло максимальной величины); если прогноз был неверным, из значения счетчика вычитается единица.
- • Если значение счетчика меньше, чем 2^{n-1}, то переход прогнозируется как невыполняемый. Если направление перехода предсказано правильно, из значения счетчика вычитается единица (если только не достигнуто значение 0); если прогноз был неверным, к значению счетчика добавляется единица.

Исследования n-битовых схем прогнозирования показали, что двухбитовая схема работает почти также хорошо, и поэтому в большинстве систем применяются двухбитовые схемы прогноза, а не n-битовые.

Буфер прогнозирования переходов может быть реализован в виде небольшой специальной кэш-памяти, доступ к которой осуществляется с помощью адреса команды во время стадии выборки команды в конвейере (IF), или как пара битов, связанных с каждым блоком кэш-памяти команд и выбираемых с каждой командой. Если команда декодируется как команда перехода, и если переход спрогнозирован как выполняемый, выборка команд начинается с целевого адреса, как только станет известным новое значение счетчика команд. В противном случае продолжается последовательная выборка и выполнение команд. Если прогноз оказался неверным, значение битов прогноза меняется в соответствии с рис. 6.8. Хотя эта схема полезна для большинства конвейеров, рассмотренный нами простейший конвейер выясняет примерно за одно и то же время оба вопроса: является ли переход выполняемым и каков целевой адрес перехода (предполагается отсутствие конфликта при обращении к регистру, определенному в команде условного перехода. Напомним, что для простейшего конвейера это справедливо, поскольку условный переход выполняет сравнение содержимого регистра с нулем во время стадии ID, во время которой вычисляется также и эффективный адрес). Таким образом, эта схема не помогает в случае простых конвейеров, подобных рассмотренному ранее.

Как уже упоминалось, точность двухбитовой схемы прогнозирования зависит от того, насколько часто прогноз каждого перехода является правильным и насколько часто строка в буфере прогнозирования соответствует выполняемой команде перехода. Если строка не соответствует данной команде перехода, прогноз в любом случае делается, поскольку все равно никакая другая информация не доступна. Даже если эта строка соответствует совсем другой команде перехода, прогноз может быть удачным.

Какую точность можно ожидать от буфера прогнозирования переходов на реальных приложениях при использовании 2 бит на каждую строку буфера? Для набора оценочных тестов SPEC-89 буфер прогнозирования переходов с 4096 строками дает точность прогноза от 99% до 82%, т.е. процент неудачных прогнозов составляет от 1% до 18% (см. рис. 6.9). Следует отметить, что буфер емкостью 4К строк считается очень большим. Буферы меньшего объема дадут худшие результаты.

Однако одного знания точности прогноза не достаточно для того, чтобы определить воздействие переходов на производительность машины, даже если известны время выполнения перехода и потери при неудачном прогнозе. Необходимо учитывать частоту переходов в программе, поскольку важность правильного прогноза больше в программах с большей частотой переходов. Например, целочисленные программы li, eqntott, expresso и gcc имеют большую частоту переходов, чем значительно более простые для прогнозирования программы плавающей точки nasa7, matrix300 и tomcatv.

Поскольку главной задачей является использование максимально доступной степени параллелизма программы, точность прогноза направления переходов становится очень важной. Как видно из рис. 6.9, точность схемы прогнозирования для целочисленных программ, которые обычно имеют более высокую частоту переходов, меньше, чем для научных программ с плавающей точкой, в которых интенсивно используются циклы. Можно решать эту проблему двумя способами: увеличением размера буфера и увеличением точности схемы, которая используется для выполнения каждого отдельного прогноза. Буфер с 4К строками уже достаточно большой и, как показывает рис. 6.9, работает практически также, что и буфер бесконечного размера. Из этого рисунка становится также ясно, что коэффициент попаданий буфера не является лимитирующим фактором. Как мы упоминали выше, увеличение числа бит в схеме прогноза также имеет малый эффект.

Рассмотренные двухбитовые схемы прогнозирования используют информацию о недавнем поведении команды условного перехода для прогноза будущего поведения этой команды. Вероятно можно улучшить точность прогноза, если учитывать не только поведение того перехода, который мы пытаемся предсказать, но рассматривать также и недавнее поведение других команд перехода. Рассмотрим, например, небольшой фрагмент из текста программы eqntott тестового пакета SPEC92 (это наихудший случай для двухбитовой схемы прогноза):

```
If  (aa==2)
        aa=0;
if  (bb==2)
        bb=0;
if  (aa!=bb) {
```

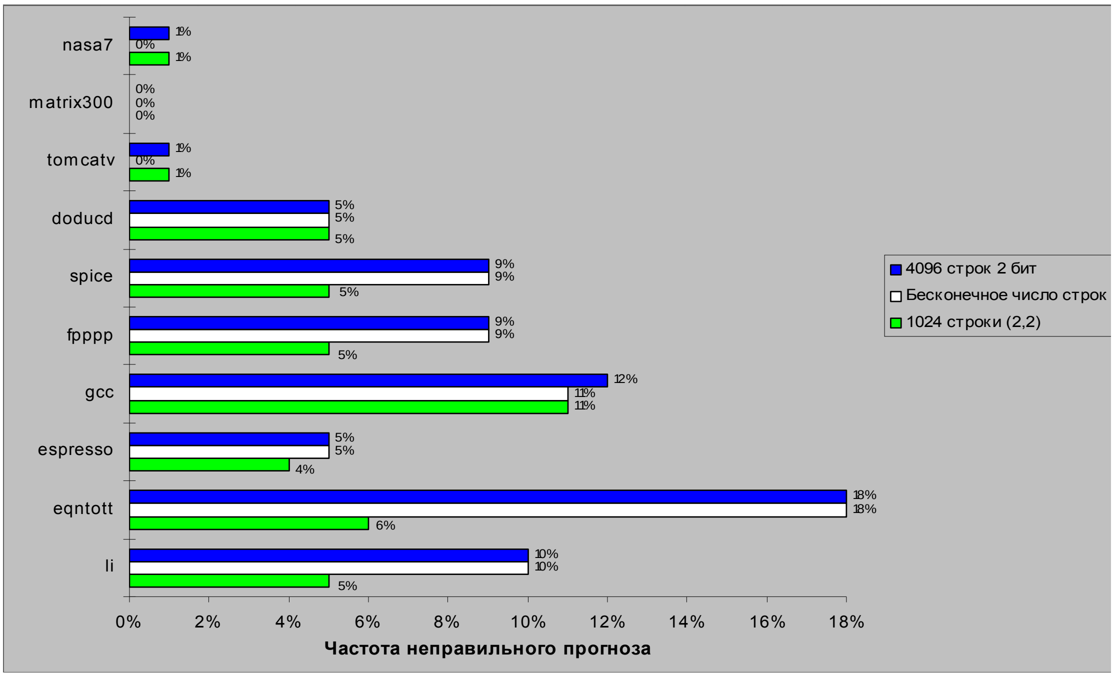


Рис. 6.9. Сравнение качества 2-битового прогноза

Ниже приведен текст сгенерированной программы (предполагается, что aa и bb размещены в регистрах R1 и R2):

```
        SUBI  R3,R1,#2
        BNEZ  R3,L1        ; переход b1 (aa!=2)
        ADD   R1,R0,R0     ; aa=0
L1:     SUBI  R3,R2,#2
        BNEZ  R3,L2        ; переход b2 (bb!=2)
        ADD   R2,R0,R0     ; bb=0
L2:     SUB   R3,R1,R2     ; R3=aa-bb
        BEQZ  R3,L3        ; branch b3  (aa==bb).
        ...
L3:
```

Пометим команды перехода как b1, b2 и b3. Можно заметить, что поведение перехода b3 коррелирует с переходами b1 и b2. Ясно, что если оба перехода b1 и b2 являются невыполняемыми (т.е. оба условия if оцениваются как истинные, и обеим переменным aa и bb присвоено значение 0), то переход b3 будет выполняемым, поскольку aa и bb очевидно равны. Схема прогнозирования, которая для предсказания направления перехода использует только прошлое поведение того же перехода, никогда этого не учтет.

Схемы прогнозирования, которые для предсказания направления перехода используют поведение других команд перехода, называются коррелированными или двухуровневыми схемами прогнозирования. Схема прогнозирования называется прогнозом (1,1), если она использует поведение одного последнего перехода для выбора из пары однобитовых схем прогнозирования на каждый переход. В общем случае схема прогнозирования (m,n) использует поведение последних m переходов для выбора из 2^m схем прогнозирования, каждая из которых представляет собой n-битовую схему прогнозирования для каждого отдельного перехода. Привлекательность такого типа коррелируемых схем прогнозирования переходов заключается в том, что они могут давать больший процент успешного прогнозирования, чем обычная двухбитовая схема, и требуют очень небольшого объема дополнительной аппаратуры. Простота аппаратной схемы определяется тем, что глобальная история последних m переходов может быть записана в m-битовом сдвиговом регистре, каждый разряд которого запоминает, был ли переход выполняемым или нет. Тогда буфер прогнозирования переходов может индексироваться конкатенацией (объединением) младших разрядов адреса перехода с m-битовой глобальной историей. Например, на рис. 6.10. показана схема прогнозирования (2,2) и организация выборки битов прогноза.

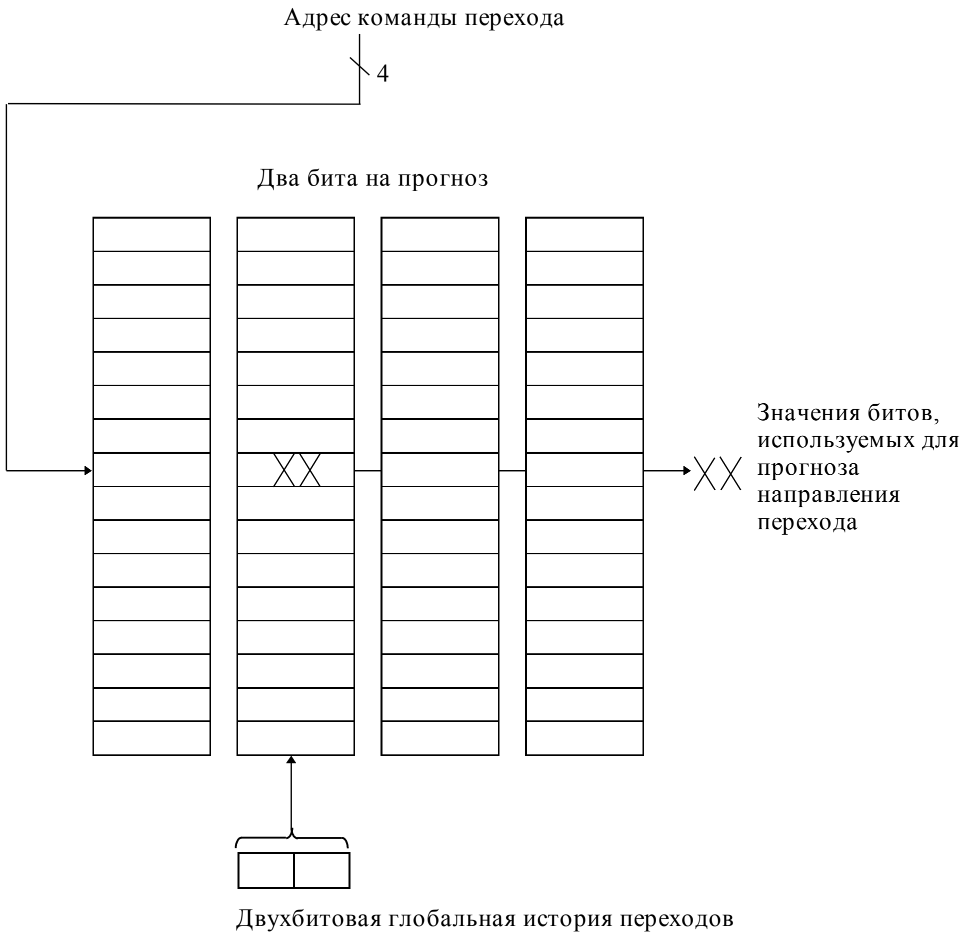


Рис. 6.10. Буфер прогнозирования переходов (2,2)

В этой реализации имеется тонкий эффект: поскольку буфер прогнозирования не является кэш-памятью, счетчики, индексируемые единственным значением глобальной схемы прогнозирования, могут в действительности в некоторый момент времени соответствовать разным командам перехода; это не отличается от того, что мы видели и раньше: прогноз может не соответствовать текущему переходу. На рис. 6.10 с целью упрощения понимания буфер изображен как двумерный объект. В действительности он может быть реализован просто как линейный массив двухбитовой памяти; индексация выполняется путем конкатенации битов глобальной истории и соответствующим числом бит, требуемых от адреса перехода. Например, на рис. 6.9 в буфере (2,2) с общим числом строк, равным 64, четыре младших разряда адреса команды перехода и два бита глобальной истории формируют 6-битовый индекс, который может использоваться для обращения к 64 счетчикам.

Насколько лучше схемы с корреляцией переходов работают по сравнению со стандартной духбитовой схемой? Чтобы их справедливо сравнить, нужно сопоставить схемы прогнозирования, использующие одинаковое число бит состояния. Число бит в схеме прогнозирования (m,n) равно $2^m \times n \times$ количество строк, выбираемых с помощью адреса перехода.

Например, двухбитовая схема прогнозирования без глобальной истории есть просто схема (0,2). Сколько бит требуется для реализации схемы прогнозирования (0,2), которую мы рассматривали раньше? Сколько бит используется в схеме прогнозирования, показанной на рис. 6.10?

Раньше мы рассматривали схему прогнозирования с 4К строками, выбираемыми адресом перехода. Таким образом, общее количество бит равно: $2^0 \times 2 \times 4K = 8K$.

Схема на рис. 6.10. имеет $2^2 \times 2 \times 16 = 128$ бит.

Чтобы сравнить производительность схемы коррелированного прогнозирования с простой двухбитовой схемой прогнозирования, производительность которой была представлена на рис. 6.8, нужно определить количество строк в схеме коррелированного прогнозирования.

Таким образом, мы должны определить количество строк, выбираемых командой перехода в схеме прогнозирования (2,2), которая содержит 8К бит в буфере прогнозирования.

Мы знаем, что

$$2^2 \times 2 \times \text{количество строк, выбираемых командой перехода} = 8K$$

Поэтому

$$\text{Количество строк, выбираемых командой перехода} = 1K.$$

На рис. 6.9 представлены результаты для сравнения простой двухбитовой схемы прогнозирования с 4К строками и схемы прогнозирования (2,2) с 1К строками. Как можно видеть, эта последняя схема прогнозирования не только превосходит простую двухбитовую схему прогнозирования с тем же самым количеством бит состояния, но часто превосходит даже двухбитовую схему прогнозирования с неограниченным (бесконечным) количеством строк. Имеется широкий спектр корреляционных схем прогнозирования, среди которых схемы (0,2) и (2,2) являются наиболее интересными.

Дальнейшее уменьшение приостановок по управлению: буфера целевых адресов переходов

Рассмотрим ситуацию, при которой на стадии выборки команд находится команда перехода (на следующей стадии будет осуществляться ее дешифрация). Тогда чтобы сократить потери, необходимо знать, по какому адресу выбирать следующую команду. Это означает, что нам как-то надо выяснить, что еще недешифрированная команда в самом деле является командой перехода, и чему равно следующее значение счетчика адресов команд. Если все это мы будем знать, то потери на команду перехода могут быть сведены к нулю. Специальный аппаратный кэш прогнозирования переходов, который хранит прогнозируемый адрес следующей команды, называется буфером целевых адресов переходов (branch-target buffer).

Каждая строка этого буфера включает программный адрес команды перехода, прогнозируемый адрес следующей команды и предысторию команды перехода (рис. 6.11). Биты предыстории представляют собой информацию о выполнении или невыполнении условий перехода данной команды в прошлом. Обращение к буферу целевых адресов перехода (сравнение с полями программных адресов команд перехода) производится с помощью текущего значения счетчика команд на этапе выборки очередной команды. Если обнаружено совпадение (попадание в терминах кэш-памяти), то по предыстории команды прогнозируется выполнение или невыполнение условий команды перехода, и немедленно производится выборка и дешифрация команд из прогнозируемой ветви программы. Считается, что предыстория перехода, содержащая информацию о двух предшествующих случаях выполнения этой команды, позволяет прогнозировать развитие событий с вполне достаточной вероятностью.

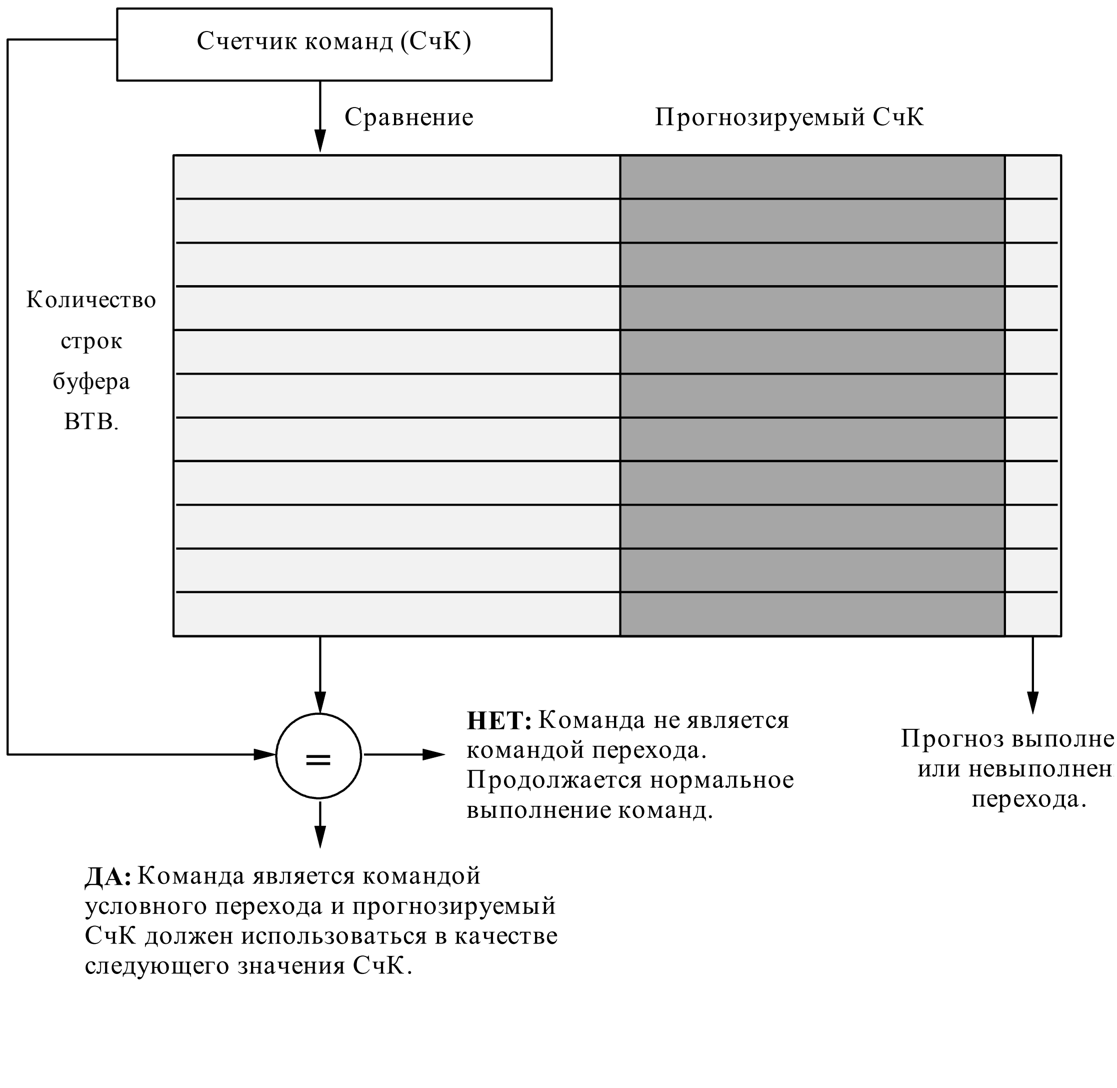


Рис. 6.11. Буфер целевых адресов переходов

Существуют и некоторые вариации этого метода. Основной их смысл заключается в том, чтобы хранить в процессоре одну или несколько команд из прогнозируемой ветви перехода. Этот метод может применяться как в совокупности с буфером целевых адресов перехода, так и без него, и имеет два преимущества. Во-первых, он позволяет выполнять обращения к буферу целевых адресов перехода в течение более длительного времени, а не только в течение времени последовательной выборки команд. Это позволяет реализовать буфер большего объема. Во-вторых, буферизация самих целевых команд позволяет использовать дополнительный метод оптимизации, который называется свертыванием переходов (branch folding). Свертывание переходов может использоваться для реализации нулевого времени выполнения самих команд безусловного перехода, а в некоторых случаях и нулевого времени выполнения условных переходов. Рассмотрим буфер целевых адресов перехода, который буферизует команды из прогнозируемой ветви. Пусть к нему выполняется обращение по адресу команды безусловного перехода. Единственной задачей этой команды безусловного перехода является замена текущего значения счетчика команд. В этом случае, когда буфер адресов регистрирует попадание и показывает, что переход безусловный, конвейер просто может заменить команду, которая выбирается из кэш-памяти (это и есть сама команда безусловного перехода), на команду из буфера. В некоторых случаях таким образом удается убрать потери для команд условного перехода, если код условия установлен заранее.

Еще одним методом уменьшения потерь на переходы является метод прогнозирования косвенных переходов, а именно переходов, адрес назначения которых меняется в процессе выполнения программы (в run-time). Компиляторы языков высокого уровня будут генерировать такие переходы для реализации косвенного вызова процедур, операторов select или case и вычисляемых операторов goto в Фортране. Однако подавляющее большинство косвенных переходов возникает в процессе выполнения программы при организации возврата из процедур. Например, для тестовых пакетов SPEC возвраты из процедур в среднем составляют 85% общего числа косвенных переходов.

Хотя возвраты из процедур могут прогнозироваться с помощью буфера целевых адресов переходов, точность такого метода прогнозирования может оказаться низкой, если процедура вызывается из нескольких мест программы или вызовы процедуры из одного места программы не локализуются по времени. Чтобы преодолеть эту проблему, была предложена концепция небольшого буфера адресов возврата, работающего как стек. Эта структура кэширует последние адреса возврата: во время вызова процедуры адрес возврата вталкивается в стек, а во время возврата он оттуда извлекается. Если этот кэш достаточно большой (например, настолько большой, чтобы обеспечить максимальную глубину вложенности вызовов), он будет прекрасно прогнозировать возвраты. На рис. 6.12 показано исполнение такого буфера возвратов, содержащего от 1 до 16 строк (элементов) для нескольких тестов SPEC.

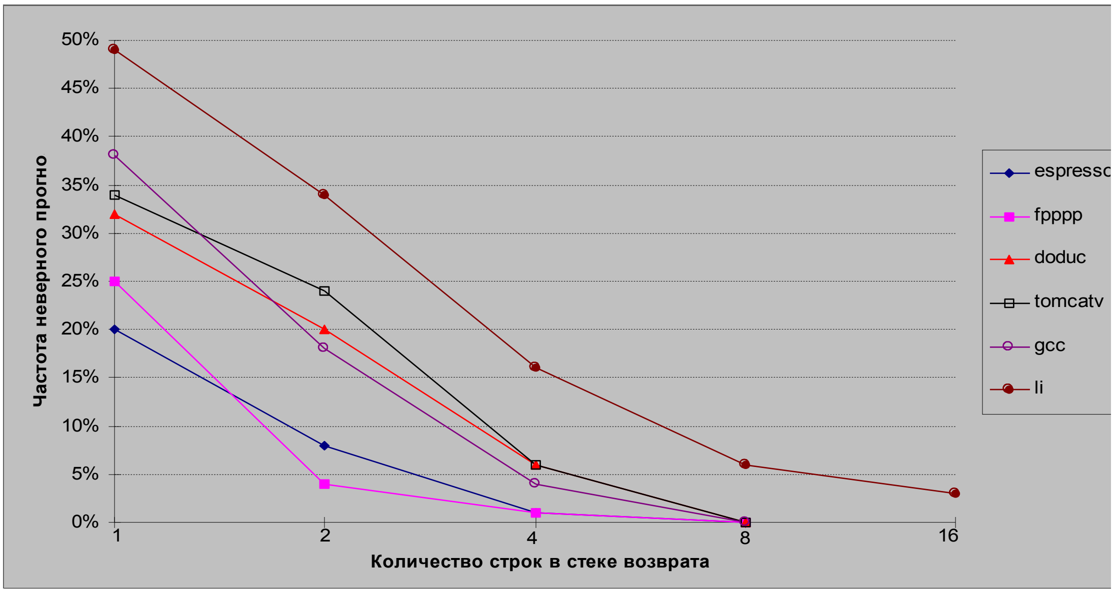


Рис. 6.12. Точность прогноза для адресов возврата

Точность прогноза в данном случае есть доля адресов возврата, предсказанных правильно. Поскольку глубина вызовов процедур обычно не большая, за некоторыми исключениями даже небольшой буфер работает достаточно хорошо. В среднем возвраты составляют 81% общего числа косвенных переходов для этих шести тестов.

Схемы прогнозирования условных переходов ограничены как точностью прогноза, так и потерями в случае неправильного прогноза. Как мы видели, типичные схемы прогнозирования достигают точности прогноза в диапазоне от 80 до 95% в зависимости от типа программы и размера буфера. Кроме увеличения точности схемы прогнозирования, можно пытаться уменьшить потери при неверном прогнозе. Обычно это делается путем выборки команд по обоим ветвям (по предсказанному и по непредсказанному направлению). Это требует, чтобы система памяти была двухпортовой, включала кэш-память с расслоением, или осуществляла выборку по одному из направлений, а затем по другому (как это делается в IBM POWER-2). Хотя подобная организация увеличивает стоимость системы, возможно это единственный способ снижения потерь на условные переходы ниже определенного уровня. Другое альтернативное решение, которое используется в некоторых машинах, заключается в кэшировании адресов или команд из нескольких направлений (ветвей) в целевом буфере.

5.4. **6.4. Одновременная выдача нескольких команд для выполнения и динамическое планирование**

Методы минимизации приостановок работы конвейера из-за наличия в программах логических зависимостей по данным и по управлению, рассмотренные в предыдущих разделах, были нацелены на достижение идеального CPI (среднего количества тактов на выполнение команды в конвейере), равного 1. Чтобы еще больше повысить производительность процессора необходимо сделать CPI меньшим, чем 1. Однако этого нельзя добиться, если в одном такте выдается на выполнение только одна команда. Следовательно необходима параллельная выдача нескольких команд в каждом такте. Существуют два типа подобного рода машин: суперскалярные машины и VLIW-машины (машины с очень длинным командным словом). Суперскалярные машины могут выдавать на выполнение в каждом такте переменное число команд, и работа их конвейеров может планироваться как статически с помощью компилятора, так и с помощью аппаратных средств динамической оптимизации. В отличие от суперскалярных машин, VLIW-машины выдают на выполнение фиксированное количество команд, которые сформатированы либо как одна большая команда, либо как пакет команд фиксированного формата. Планирование работы VLIW-машины всегда осуществляется компилятором.

Суперскалярные машины используют параллелизм на уровне команд путем посылки нескольких команд из обычного потока команд в несколько функциональных устройств. Дополнительно, чтобы снять ограничения последовательного выполнения команд, эти машины используют механизмы внеочередной выдачи и внеочередного завершения команд, прогнозирование переходов, кэши целевых адресов переходов и условное (по предположению) выполнение команд. Возросшая сложность, реализуемая этими механизмами, создает также проблемы реализации точного прерывания.

В типичной суперскалярной машине аппаратура может осуществлять выдачу от одной до восьми команд в одном такте. Обычно эти команды должны быть независимыми и удовлетворять некоторым ограничениям, например таким, что в каждом такте не может выдаваться более одной команды обращения к памяти. Если какая-либо команда в потоке команд является логически зависимой или не удовлетворяет критериям выдачи, на выполнение будут выданы только команды, предшествующие данной. Поэтому скорость выдачи команд в суперскалярных машинах является переменной. Это отличает их от VLIW-машин, в которых полную ответственность за формирование пакета команд,

которые могут выдаваться одновременно, несет компилятор, а аппаратура в динамике не принимает никаких решений относительно выдачи нескольких команд.
Предположим, что машина может выдавать на выполнение две команды в одном такте. Одной из таких команд может быть команда загрузки регистров из памяти, записи регистров в память, команда переходов, операции целочисленного АЛУ, а другой может быть любая операция плавающей точки. Параллельная выдача целочисленной операции и операции с плавающей точкой намного проще, чем выдача двух произвольных команд. В реальных системах (например, в микропроцессорах PA7100, hyperSPARC, Pentium и др.) применяется именно такой подход. В более мощных микропроцессорах (например, MIPS R10000, UltraSPARC, PowerPC 620 и др.) реализована выдача до четырех команд в одном такте.
Выдача двух команд в каждом такте требует одновременной выборки и декодирования, по крайней мере, 64 бит. Чтобы упростить декодирование можно потребовать, чтобы команды располагались в памяти парами и были выровнены по 64-битовым границам. В противном случае необходимо анализировать команды в процессе выборки и, возможно, менять их местами в момент пересылки в целочисленное устройство и в устройство ПТ. При этом возникают дополнительные требования к схемам обнаружения конфликтов. В любом случае вторая команда может выдаваться, только если может быть выдана на выполнение первая команда. Аппаратура принимает такие решения в динамике, обеспечивая выдачу только первой команды, если условия для одновременной выдачи двух команд не соблюдаются. На рис. 6.13 представлена диаграмма работы подобного конвейера в идеальном случае, когда в каждом такте на выполнение выдается пара команд.

Тип команды	**Ступень конвейера**							
Целочисленная команда	IF	ID	EX	MEM	WB			
Команда ПТ	IF	ID	EX	MEM	WB			
Целочисленная команда		IF	ID	EX	MEM	WB		
КомандаПТ		IF	ID	EX	MEM	WB		
Целочисленная команда			IF	ID	EX	MEM	WB	
Команда ПТ			IF	ID	EX	MEM	WB	
Целочисленная команда				IF	ID	EX	MEM	WB
Команда ПТ				IF	ID	EX	MEM	WB

Рис. 6.13. Работа суперскалярного конвейера

Такой конвейер позволяет существенно увеличить скорость выдачи команд. Однако чтобы он смог так работать, необходимо иметь либо полностью конвейеризованные устройства плавающей точки, либо соответствующее число независимых функциональных устройств. В противном случае устройство плавающей точки станет узким горлом и эффект, достигнутый за счет выдачи в каждом такте пары команд, сведется к минимуму.
При параллельной выдаче двух операций (одной целочисленной команды и одной команды ПТ) потребность в дополнительной аппаратуре, помимо обычной логики обнаружения конфликтов, минимальна: целочисленные операции и операции ПТ используют разные наборы регистров и разные функциональные устройства. Более того, усиление ограничений на выдачу команд, которые можно рассматривать как специфические структурные конфликты (поскольку выдаваться на выполнение могут

только определенные пары команд), обнаружение которых требует только анализа кодов операций. Единственная сложность возникает, только если команды представляют собой команды загрузки, записи и пересылки чисел с плавающей точкой. Эти команды создают конфликты по портам регистров ПТ, а также могут приводить к новым конфликтам типа RAW, когда операция ПТ, которая могла бы быть выдана в том же такте, является зависимой от первой команды в паре.

Проблема регистровых портов может быть решена, например, путем реализации отдельной выдачи команд загрузки, записи и пересылки с ПТ. В случае составления ими пары с обычной операцией ПТ ситуацию можно рассматривать как структурный конфликт. Такую схему легко реализовать, но она будет иметь существенное воздействие на общую производительность. Конфликт подобного типа может быть устранен посредством реализации в регистровом файле двух дополнительных портов (для выборки и записи).

Если пара команд состоит из одной команды загрузки с ПТ и одной операции с ПТ, которая от нее зависит, необходимо обнаруживать подобный конфликт и блокировать выдачу операции с ПТ. За исключением этого случая, все другие конфликты естественно могут возникать, как и в обычной машине, обеспечивающей выдачу одной команды в каждом такте. Для предотвращения ненужных приостановок могут, правда, потребоваться дополнительные цепи обхода.

Другой проблемой, которая может ограничить эффективность суперскалярной обработки, является задержка загрузки данных из памяти. В нашем примере простого конвейера команды загрузки имели задержку в один такт, что не позволяло следующей команде воспользоваться результатом команды загрузки без приостановки. В суперскалярном конвейере результат команды загрузки не может быть использован в том же самом и в следующем такте. Это означает, что следующие три команды не могут использовать результат команды загрузки без приостановки. Задержка перехода также становится длиною в три команды, поскольку команда перехода должна быть первой в паре команд. Чтобы эффективно использовать параллелизм, доступный на суперскалярной машине, нужны более сложные методы планирования потока команд, используемые компилятором или аппаратными средствами, а также более сложные схемы декодирования команд.

Рассмотрим, например, что дает разворачивание циклов и планирование потока команд для суперскалярного конвейера. Ниже представлен цикл, который мы уже разворачивали и планировали его выполнение на простом конвейере.

```
Loop: LD   F0,0(R1)     ;F0=элемент вектора
      ADDD F4,F0,F2     ;добавление скалярной величины из F2
      SD   0(R1),F4     ;запись результата
      SUBI R1,R1,#8     ;декрементирование указателя
                        ;8 байт на двойное слово
      BNEZ R1,Loop      ;переход R1!=нулю
```

Чтобы спланировать этот цикл для работы без задержек, необходимо его развернуть и сделать пять копий тела цикла. После такого разворачивания цикл будет содержать по пять команд LD, ADDD, и SD, а также одну команду SUBI и один условный переход BNEZ. Развернутая и оптимизированная программа этого цикла дана ниже:

Целочисленная команда	Команда ПТ	Номер такта
Loop: LD F0,0(R1)		1
LD F8,-8(R1)		2
LD F10,-16(R1)	ADDD F4,F0,F2	3
LD F14,-24(R1)	ADDD F8,F6,F2	4
LD F18,-32(R1)	ADDD F12,F10,F2	5

`SD 0(R1),F4`	ADDD F16,F14,F2	6
`SD -8(R1),F8`	ADDD F20,F18,F2	7
`SD -16(R1),F12`		8
`SD -24(R1),F16`		9
`SUBI R1,R1,#40`		10
`BNEZ R1,Loop`		11
`SD -32(R1),F20`		12

Этот развернутый суперскалярный цикл теперь работает со скоростью 12 тактов на итерацию, или 2.4 такта на один элемент (по сравнению с 3.5 тактами для оптимизированного развернутого цикла на обычном конвейере). В этом примере производительность суперскалярного конвейера ограничена существующим соотношением целочисленных операций и операций ПТ, но команд ПТ не достаточно для поддержания полной загрузки конвейера ПТ. Первоначальный оптимизированный неразвернутый цикл выполнялся со скоростью 6 тактов на итерацию, вычисляющую один элемент. Мы получили, таким образом, ускорение в 2.5 раза, больше половины которого произошло за счет разворачивания цикла. Чистое ускорение за счет суперскалярной обработки дало улучшение примерно в 1.5 раза.

В лучшем случае такой суперскалярный конвейер позволит выбирать две команды и выдавать их на выполнение, если первая из них является целочисленной, а вторая - с плавающей точкой. Если это условие не соблюдается, что легко проверить, то команды выдаются последовательно. Это показывает два главных преимущества суперскалярной машины по сравнению с VLIW-машиной. Во-первых, малое воздействие на плотность кода, поскольку машина сама определяет, может ли быть выдана следующая команда, и нам не надо следить за тем, чтобы команды соответствовали возможностям выдачи. Во-вторых, на таких машинах могут работать неоптимизированные программы, или программы, откомпилированные в расчете на более старую реализацию. Конечно, такие программы не могут работать очень хорошо. Один из способов улучшить ситуацию заключается в использовании аппаратных средств динамической оптимизации.

В общем случае в суперскалярной системе команды могут выполняться параллельно и возможно не в порядке, предписанном программой. Если не предпринимать никаких мер, такое неупорядоченное выполнение команд и наличие множества функциональных устройств с разными временами выполнения операций могут приводить к дополнительным трудностям. Например, при выполнении некоторой длинной команды с плавающей точкой (команды деления или вычисления квадратного корня) может возникнуть исключительная ситуация уже после того, как завершилось выполнение более быстрой операции, выданной после этой длинной команды. Для того, чтобы поддерживать модель точных прерываний, аппаратура должна гарантировать корректное состояние процессора при прерывании для организации последующего возврата.

Обычно в машинах с неупорядоченным выполнением команд предусматриваются дополнительные буферные схемы, гарантирующие завершение выполнения команд в строгом порядке, предписанном программой. Такие схемы представляют собой некоторый буфер "истории", т.е. аппаратную очередь, в которую при выдаче попадают команды и текущие значения регистров результата этих команд в заданном программой порядке.

В момент выдачи команды на выполнение она помещается в конец этой очереди, организованной в виде буфера FIFO (первый вошел - первый вышел). Единственный способ для команды достичь головы этой очереди - завершение выполнения всех предшествующих ей операций. При неупорядоченном выполнении некоторая команда может завершить свое выполнение, но все еще будет находиться в середине очереди.

Команда покидает очередь, когда она достигает головы очереди и ее выполнение завершается в соответствующем функциональном устройстве. Если команда находится в голове очереди, но ее выполнение в функциональном устройстве не закончено, она очередь не покидает. Такой механизм может поддерживать модель точных прерываний, поскольку вся необходимая информация хранится в буфере и позволяет скорректировать состояние процессора в любой момент времени.
Этот же буфер "истории" позволяет реализовать и условное (speculative) выполнение команд (выполнение по предположению), следующих за командами условного перехода. Это особенно важно для повышения производительности суперскалярных архитектур. Статистика показывает, что на каждые шесть обычных команд в программах приходится в среднем одна команда перехода. Если задерживать выполнение следующих за командой перехода команд, потери на конвейеризацию могут оказаться просто неприемлемыми. Например, при выдаче четырех команд в одном такте в среднем в каждом втором такте выполняется команда перехода. Механизм условного выполнения команд, следующих за командой перехода, позволяет решить эту проблему. Это условное выполнение обычно связано с последовательным выполнением команд из заранее предсказанной ветви команды перехода. Устройство управления выдает команду условного перехода, прогнозирует направление перехода и продолжает выдавать команды из этой предсказанной ветви программы.
Если прогноз оказался верным, выдача команд так и будет продолжаться без приостановок. Однако если прогноз был ошибочным, устройство управления приостанавливает выполнение условно выданных команд и, если необходимо, использует информацию из буфера истории для ликвидации всех последствий выполнения условно выданных команд. Затем начинается выборка команд из правильной ветви программы. Таким образом, аппаратура, подобная буферу, истории позволяет не только решить проблемы с реализацией точного прерывания, но и обеспечивает увеличение производительности суперскалярных архитектур.

5.5. **6.5. Архитектура машин с длинным командным словом**

Архитектура машин с очень длинным командным словом (VLIW - Very Long Instruction Word) позволяет сократить объем оборудования, требуемого для реализации параллельной выдачи нескольких команд, и потенциально чем большее количество команд выдается параллельно, тем больше эта экономия. Например, суперскалярная машина, обеспечивающая параллельную выдачу двух команд, требует параллельного анализа двух кодов операций, шести полей номеров регистров, а также того, чтобы динамически анализировалась возможность выдачи одной или двух команд и выполнялось распределение этих команд по функциональным устройствам. Хотя требования по объему аппаратуры для параллельной выдачи двух команд остаются достаточно умеренными, и можно даже увеличить степень распараллеливания до четырех (что применяется в современных микропроцессорах), дальнейшее увеличение количества выдаваемых параллельно для выполнения команд приводит к нарастанию сложности реализации из-за необходимости определения порядка следования команд и существующих между ними зависимостей.
Архитектура VLIW базируется на множестве независимых функциональных устройств. Вместо того, чтобы пытаться параллельно выдавать в эти устройства независимые команды, в таких машинах несколько операций упаковываются в одну очень длинную команду. При этом ответственность за выбор параллельно выдаваемых для выполнения операций полностью ложится на компилятор, а аппаратные средства, необходимые для реализации суперскалярной обработки, просто отсутствуют.

VLIW-команда может включать, например, две целочисленные операции, две операции с плавающей точкой, две операции обращения к памяти и операцию перехода. Такая команда будет иметь набор полей для каждого функционального устройства, возможно от 16 до 24 бит на устройство, что приводит к команде длиною от 112 до 168 бит.
Рассмотрим работу цикла инкрементирования элементов вектора на подобного рода машине в предположении, что одновременно могут выдаваться две операции обращения к памяти, две операции с плавающей точкой и одна целочисленная операция либо одна команда перехода. На рис. 6.14 показан код для реализации этого цикла. Цикл был развернут семь раз, что позволило устранить все возможные приостановки конвейера. Один проход по циклу осуществляется за 9 тактов и вырабатывает 7 результатов. Таким образом, на вычисление каждого результата расходуется 1.28 такта (в нашем примере для суперскалярной машины на вычисление каждого результата расходовалось 2.4 такта).
Для машин с VLIW-архитектурой был разработан новый метод планирования выдачи команд, названный "трассировочным планированием". При использовании этого метода из последовательности исходной программы генерируются длинные команды путем просмотра программы за пределами базовых блоков. Как уже отмечалось, базовый блок - это линейный участок программы без ветвлений.
С точки зрения архитектурных идей машину с очень длинным командным словом можно рассматривать как расширение RISC-архитектуры. Как и в RISC-архитектуре, аппаратные ресурсы VLIW-машины предоставлены компилятору, и ресурсы планируются статически. В машинах с очень длинным командным словом к этим ресурсам относятся конвейерные функциональные устройства, шины и банки памяти. Для поддержки высокой пропускной способности между функциональными устройствами и регистрами необходимо использовать несколько наборов регистров. Аппаратное разрешение конфликтов исключается, и предпочтение отдается простой логике управления. В отличие от традиционных машин регистры и шины не резервируются, а их использование полностью определяется во время компиляции.

Обращение к памяти 1	Обращение к памяти 2	Операция ПТ 1	Операция ПТ 2	Целочисленная операция/ переход
LD F0,0(R1)	LD F6,-8(R1)			
LD F10,-16(R1)	LD F14,-24(R1)			
LD F18,-32(R1)	LD F22,-40(R1)	ADDD F4,F0,F2	ADDD F8,F6,F2	
LD F26,-48(R1)		ADDD F12,F10,F2	ADDD F16,F14,F2	
		ADDD F20,F18,F2	ADDD F24,F22,F2	
	SD -8(R1),F8	ADDD F28,F26,F2		
SD 0(R1),F4	SD -24(R1),F16			SUBI R1,R1,#48
SD -16(R1),F12	SD -40(R1),F24			BNEZ R1,Loop
SD -32(R1),F20				
SD 0(R1),F28				

Рис. 6.14.

В машинах типа VLIW, кроме того, этот принцип замены управления во время выполнения программы планированием во время компиляции распространен на системы памяти. Для поддержания занятости конвейерных функциональных устройств должна быть обеспечена высокая пропускная способность памяти. Одним из современных подходов к увеличению пропускной способности памяти является использование расслоения памяти. Однако в системе с расслоенной памятью возникает конфликт банка, если банк занят предыдущим обращением. В обычных машинах состояние занятости банков памяти отслеживается аппаратно и проверяется, когда выдается команда, выполнение которой связано с обращением к памяти. В машине типа VLIW эта функция передана программным средствам. Возможные конфликты банков определяет специальный модуль компилятора - модуль предотвращения конфликтов.

Обнаружение конфликтов не является задачей оптимизации, это скорее функция контроля корректности выполнения операций. Компилятор должен быть способен определять, что конфликты невозможны или, в противном случае, допускать, что может возникнуть наихудшая ситуация. В определенных ситуациях, например, в том случае, когда производится обращение к массиву, а индекс вычисляется во время выполнения программы, простого решения здесь нет. Если компилятор не может определить, что конфликт не произойдет, операции не могут планироваться для параллельного выполнения, а это ведет к снижению производительности.

Компилятор с трассировочным планированием определяет участок программы без обратных дуг (переходов назад), которая становится кандидатом для составления расписания. Обратные дуги обычно имеются в программах с циклами. Для увеличения размера тела цикла широко используется методика раскрутки циклов, что приводит к образованию больших фрагментов программы, не содержащих обратных дуг. Если дана программа, содержащая только переходы вперед, компилятор делает эвристическое предсказание выбора условных ветвей. Путь, имеющий наибольшую вероятность выполнения (его называют трассой), используется для оптимизации, проводимой с учетом зависимостей по данным между командами и ограничений аппаратных ресурсов. Во время планирования генерируется длинное командное слово. Все операции длинного командного слова выдаются одновременно и выполняются параллельно.

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

Ключевым условием достижения эффективной работы VLIW-машины является корректное предсказание выбора условных ветвей. Отмечено, например, что прогноз условных ветвей для научных программ часто оказывается точным. Возвраты назад имеются во всех итерациях цикла, за исключением последней. Таким образом, "прогноз", который уже дается самими переходами назад, будет корректен в большинстве случаев. Другие условные ветви, например ветвь обработки переполнения и проверки граничных условий (выход за границы массива), также надежно предсказуемы.

5.6. 6.6. Обнаружение и устранение зависимостей компилятором и разворачивание циклов

В этом разделе мы обсудим методы компиляции, которые позволяют увеличить степень параллелизма, который можно использовать при выполнении программы. Мы начнем с изучения методов обнаружения зависимостей и устранения зависимостей по именам.

Обнаружение и устранение зависимостей

Нахождение зависимостей по данным в программе является важной частью трех задач: (1) хорошее планирование программного кода, (2) определение циклов, которые могут содержать параллелизм, и (3) устранение зависимостей по именам. Сложность анализа зависимостей связана с наличием массивов (и указателей в языках, подобных языку Си). Поскольку обращения к скалярным переменным осуществляется явно по имени, они обычно могут анализироваться достаточно просто. При этом наличие указателей-алиасов и обращений к параметрам вызывает усложнения, поскольку они могут быть неизвестны в процессе анализа.

При анализе необходимо найти все зависимости и определить, имеется ли цикл в этих зависимостях, поскольку это то, что не позволяет нам выполнять цикл параллельно. Рассмотрим следующий пример:

```
for (i=1;i<=100;i=i+1) {
        A[i] = B[i] + C[i];
        D[i] = A[i] + E[i];
                }
```

Поскольку в данном случае зависимость, связанная с A, не приводит к зависимости между итерациями цикла, можно развернуть цикл для выявления большей степени параллелизма. Мы не можем прямо поменять местами два обращения к A. Если цикл имеет зависимости между итерациями, которые не являются циклическими, можно сначала преобразовать цикл для устранения этих зависимостей, а затем развернуть цикл для выявления большей степени параллелизма. Во многих параллельных циклах степень параллелизма ограничена только количеством разворотов цикла, которое в свою очередь ограничивается только количеством итераций цикла. Конечно на практике, чтобы получить выигрыш от этой большей степени параллелизма, потребуется много функциональных устройств и огромное количество регистров. Отсутствие зависимости между итерациями цикла просто сообщает нам, что нам доступна большая степень параллелизма.

Фрагмент вышеприведенного кода иллюстрирует также другую возможность для улучшения машинного кода. Второе обращение к A не нужно транслировать в команду загрузки из памяти, поскольку мы знаем, что значение вычислено и записано предыдущим оператором. Поэтому второе обращение к A может выполняться с помощью обращения к регистру, в котором значение A было вычислено. Выполнение этой оптимизации требует знания того, что два обращения всегда относятся к одному и тому же адресу памяти, и что к той же самой ячейке между этими двумя обращениями другие обращения (по записи) отсутствуют. Обычно анализ зависимостей по данным дает информацию только о том, что одно обращение может зависеть от другого. Для определения того, что два обращения должны выполняться точно по одному и тому же адресу, требуется более сложный анализ. В вышеприведенном примере достаточно простейшей версии такого анализа, поскольку оба обращения находятся в одном и том же базовом блоке.

Часто зависимости между итерациями цикла появляются в форме рекуррентного отношения:

```
for (i=2;i<=100;i=i+1) {
        Y[i] = Y[i-1] + Y[i];
                }
```

Определение наличия рекуррентных отношений может оказаться важным по двум причинам. Некоторые архитектуры (особенно векторные машины) имеют специальную поддержку для выполнения рекуррентных отношений, и некоторые рекуррентные отношения могут быть источником значительной степени параллелизма. Например, рассмотрим цикл:

```
for (i=6;i<=100;i=i+1) {
        Y[i] = Y[i-5] + Y[i];
                }
```

На итерации j цикл обращается к элементу j-5. Говорят, что цикл имеет зависимость с расстоянием 5. Предыдущий цикл имел зависимость с расстоянием 1. Чем больше расстояние, тем больше степень потенциального параллелизма, которую можно получить при помощи разворачивания цикла. Например, если мы разворачиваем первый цикл, имеющий зависимость с расстоянием 1, последовательные операторы зависят друг от друга; имеется некоторая степень параллелизма между отдельными командами, но не очень большая. Если мы разворачиваем цикл, который имеет зависимость с расстоянием 5, то имеется последовательность пяти команд, которые не имеют зависимостей, и тем самым обладают значительно большей степенью параллелизма уровня команд. Хотя многие циклы с зависимостями между итерациями имеют расстояние зависимостей 1, случаи с большими расстояниями в действительности возникают, и большее расстояние между зависимостями может обеспечивать достаточную степень параллелизма для поддержания машины занятой.

Как вообще компилятор обнаруживает зависимости? Почти все алгоритмы анализа зависимостей работают с предположением, что обращения к массивам являются аффинными. В простейшем случае индекс одномерного массива является аффинным, если он может быть записан в форме: $a \times i + b$, где a и b константы, a i - переменная индекса цикла. Индекс многомерного массива является аффинным, если индекс по каждой размерности является аффинным.

Таким образом, определение факта наличия зависимостей между двумя обращениями к одному и тому же массиву в цикле сводится к определению того, что две аффинные функции могут иметь одно и то же значение для различных индексов между границами цикла. Например, предположим, что мы выполнили запись в элемент массива со значением индекса $a \times i + b$, и выполняем загрузку из того же массива со значением индекса $c \times i + d$, где i - переменная индекса цикла for, которая меняется в пределах от m до n. Зависимость существует, если имеют место два условия:

1. 1. Имеются индексы двух итераций, j и k, оба внутри пределов цикла for. А именно,
   $m \le j, k \le n$.
2. 2. Цикл выполняет запись в элемент массива, индексируемого при помощи $a \times j + b$, и затем выбирает значение из того же самого элемента массива, когда он индексируется с помощью $c \times k + d$. А именно, $a \times j + b = c \times k + d$.

В общем случае во время компиляции мы не можем определить, имеет ли место зависимость. Например, значения a, b, c и d могут быть неизвестными (они могут быть значениями другого массива), а, следовательно, невозможно сказать, что зависимость существует. В других случаях проверка зависимостей может оказаться очень дорогой, но в принципе возможной во время компиляции. Например, обращения могут зависеть от индексов итераций множества вложенных циклов. Однако многие программы содержат в основном простые индексы, где a, b, c и d все являются константами. Для этих случаев можно придумать недорогие тесты для обнаружения зависимостей.

Например, простым и достаточным тестом отсутствия зависимостей является наибольший общий делитель, или тест НОД. Он основан на том, что если существует зависимость между итерациями цикла, то НОД(c,a) должен делить (d-b). (Вспомним, что целое x делит другое целое y, если отсутствует остаток от деления y/x). Тест НОД является достаточным, чтобы гарантировать, что зависимости отсутствуют. Однако имеются случаи, когда тест НОД достигает цели, но в реальной программе зависимость отсутствует. Это может возникнуть, например, поскольку тест НОД не рассматривает границы цикла.

В общем случае задача определения действительного наличия зависимостей является NP-полной. Однако на практике многие частые случаи могут быть точно проанализированы при вполне умеренных затратах. Тест является точным, если он точно определяет наличие зависимости. Хотя общий случай является NP-полным (т.е. точное решение можно найти путем полного перебора всех вариантов), имеются точные тесты для ограниченного числа ситуаций, которые являются намного более дешевыми.

Кроме определения наличия зависимостей, компилятор старается также классифицировать тип зависимости. Это позволяет компилятору распознать зависимости по именам и устранить их путем переименования и копирования.

Например, следующий цикл имеет несколько типов зависимостей. Попробуем найти все истинные зависимости, зависимости по выходу и антизависимости и устранить зависимости по выходу и антизависимости с помощью переименования.

```
for (i=1;i<=100;i=i+1) {
        Y[i] = X[i] / c; /*S1*/
        X[i] = X[i] + c; /*S2*/
        Z[i] = Y[i] + c; /*S3*/
        Y[i] = c - Y[i]; /*S4*/
                        }
```

В этих четырех операторах имеются следующие зависимости:

1. 1. Имеются истинные зависимости от S1 к S3 и от S1 к S4 из-за Y[i]. Отсутствует зависимость между итерациями цикла, что позволяет рассматривать цикл как параллельный. Эти зависимости приведут к ожиданию операторами S3 и S4 завершения оператора S1.
2. 2. Имеется антизависимость от S1 к S2.
3. 3. Имеется зависимость по выходу от S1 к S4.

Следующая версия цикла устраняет эти ложные (или псевдо-) зависимости.

```
for (i=1;i<=100;i=i+1) {
                        /* Y переименовывается в T для устранения
                        зависимости по выходу */
        T[i] = X[i] / c;
                        /* X переименовывается в X1 для устранения
                        антизависимости */
        X1[i] = X[i] + c;
        Z[i] = T[i] + c;
        Y[i] = c - T[i];
                        }
```

После цикла переменная X оказалась переименованной в X1. В коде программы, следующем за циклом, компилятор просто может заменить имя X на имя X1. В данном случае переименование не требует действительной операции копирования, а может быть выполнено с помощью заменяющего имени или соответствующего распределения регистров. Однако в других случаях переименование может потребовать копирования.

Анализ зависимостей является важнейшей технологией для улучшения использования параллелизма. На уровне команд она дает информацию, необходимую для изменения в процессе планирования порядка обращений к памяти, а также для определения полезности разворачивания цикла. Для обнаружения параллелизма уровня цикла анализ зависимостей является базовым инструментом. Эффективная компиляция программ для векторных машин, а также для мультипроцессоров существенно зависит от этого анализа. Кроме того, при планировании потока команд полезно определить, являются ли потенциально зависимыми обращения к памяти. Главный недостаток анализа зависимостей заключается в том, что он применим при ограниченном наборе обстоятельств, а именно к обращениям внутри одного гнезда циклов и использует аффинные функции индексов. Таким образом, имеется огромное многообразие ситуаций, при которых анализ зависимостей не может сообщить нам то, что мы хотели бы знать, а именно:

- • когда обращения к объектам выполняются с помощью указателей, а не индексов массива,
- • когда индексация массива осуществляется косвенно через другой массив, что имеет место при работе с разреженными массивами,

- • зависимость может существовать для некоторого значения входов, но отсутствовать в действительности при выполнении программы, поскольку входы никогда не принимают определенных значений,
- • когда оптимизация зависит не просто от знания возможности наличия зависимости, но требует точного определения того, от какой операции записи зависит чтение переменной.

Программная конвейеризация: символическое разворачивание циклов

Мы уже видели, что один из методов компиляции - разворачивание циклов - полезен для увеличения степени параллелизма на уровне команд посредством создания более длинных последовательностей линейного кода. Имеются два других важных метода, которые разработаны для этих целей: программная конвейеризация и планирование трасс.

Программная конвейеризация - это метод реорганизации циклов таким образом, что каждая итерация в программно конвейеризованном коде составляется из команд, выбранных из разных итераций первоначального цикла. Планировщик по существу чередует команды из разных итераций цикла так, чтобы отдалить друг от друга зависимые команды, которые возникают в одной итерации цикла. Программно конвейеризованный цикл чередует команды из разных итераций без реального разворачивания цикла (рис. 6.15). Этот метод по существу программно выполняет то, что алгоритм Томасуло делает с помощью аппаратных средств. Программно конвейеризованный цикл будет содержать по одной команде, каждая из которых относится к разным итерациям. Для начального запуска цикла (пролог цикла) и для завершения цикла (эпилог цикла) требуются некоторые команды.

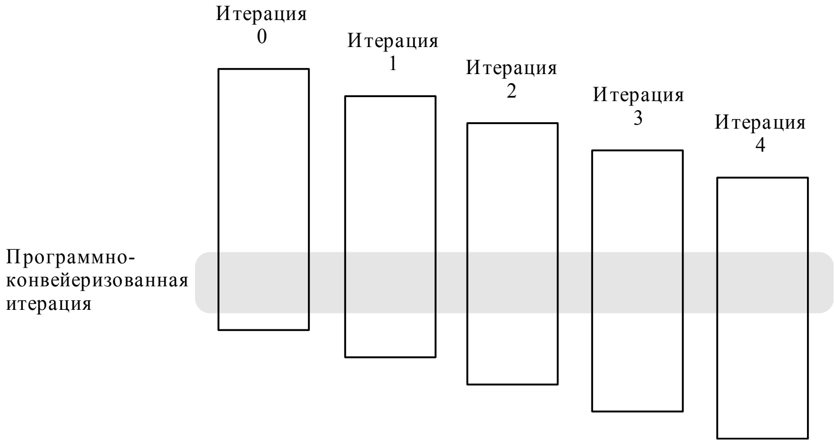


Рис. 6.15. Программная конвейеризация

Например, рассмотрим программно конвейеризованную версию нижеприведенного цикла, который складывает с содержимым регистра F2 все элементы некоторого массива с начальным адресом, хранящимся в регистре R1.

```
Loop:  LD    F0,0(R1)
       ADDD  F4,F0,F2
       SD    0(R1),F4
       SUBI  R1,R1,#8
       BNEZ  R1, Loop
```

Игнорируя пролог и эпилог, мы можем переписать цикл следующим образом:

```
Loop:  SD    0(R1),F4        ; записывает в M[i]
```

```
ADDD F4,F0,F2      ; складывает с M[i-1]
LD   F0,-16(R1)    ; загружает M[i-2]
BNEZ R1, Loop
SUBI R1,R1,#8      ; вычитает в слоте задержки
```

Если не принимать во внимание пролог и эпилог, этот цикл может работать со скоростью 5 тактов на один проход. Поскольку команда загрузки осуществляет выборку на расстоянии двух элементов от счетчика цикла, цикл должен выполнять на две итерации меньше. При этом перед началом цикла из содержимого регистра R1 необходимо вычесть 16. Заметим, что повторное использование регистров (например, F4, F0 и R1) требует использования специальных аппаратных средств, чтобы обойти конфликты типа WAR и приостановки конвейера. В данном случае это не должно привести к каким-либо проблемам, поскольку никаких приостановок по причине зависимостей по данным произойти не должно.

Управление регистрами в программно конвейеризуемых циклах может быть достаточно сложным. Вышеприведенный пример не слишком тяжелый, поскольку в регистры выполняется запись в одной итерации, а их чтение происходит в следующей. В других случаях может потребоваться увеличить количество итераций между моментом выдачи команды и моментом, когда используется ее результат. Это происходит, когда в теле цикла имеется небольшое количество команд, а задержки их выполнения достаточно большие. В этих случаях требуется комбинация методов программной конвейеризации и разворачивания цикла.

Программную конвейеризацию можно рассматривать как символическое разворачивание цикла. Действительно, некоторые алгоритмы программной конвейеризации используют разворачивание цикла в качестве исходного материала для расчета (вычисления) выполнения программной конвейеризации. Главное преимущество программной конвейеризации по отношению к прямому разворачиванию циклов заключается в том, что первая генерирует в результате меньший по размеру программный код. Программная конвейеризация и разворачивание циклов в дополнение к тому, что они дают лучше спланированный внутренний цикл, сами по себе сокращают разные типы накладных расходов. Разворачивание циклов сокращает накладные расходы на организацию цикла, связанные с командами перехода и изменения значения счетчика циклов. Программная конвейеризация сокращает время, когда цикл не работает с полной скоростью, что происходит только однажды в начале и в конце цикла. Если мы разворачиваем цикл, который выполняет 100 итераций постоянное количество раз, скажем 4 раза, мы будем иметь накладные расходы 100/4=25 раз - каждый раз, когда будет инициироваться внутренний развернутый цикл. На рис. 6.16 это поведение показано графически. Поскольку эти методы направлены на два различных типа накладных расходов, наилучший результат может быть получен при использовании обоих методов.

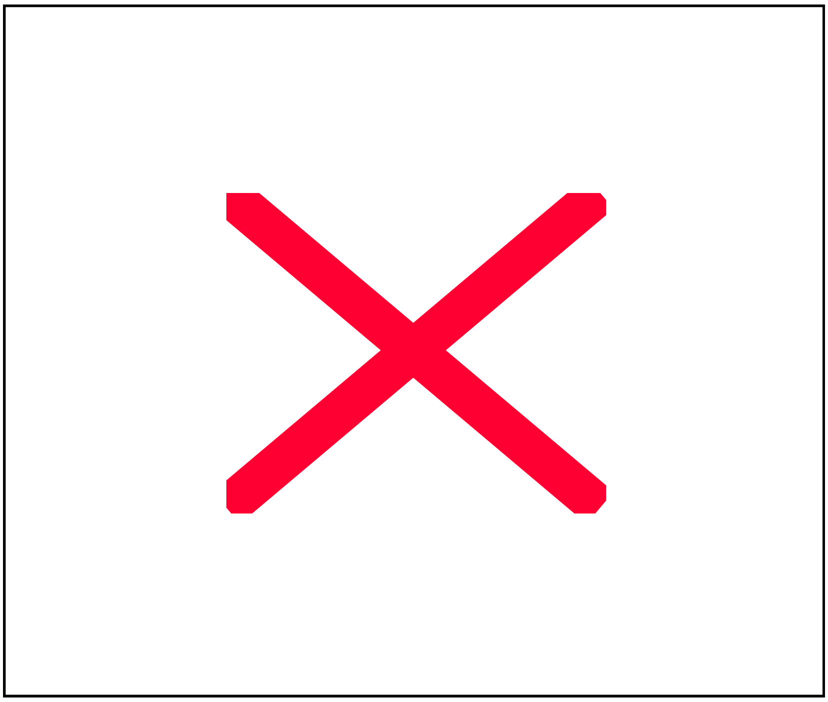


Рис. 6.16.

Другим методом, используемым для выделения дополнительного параллелизма, является *трассировочное планирование*. Трассировочное планирование расширяет метод разворачивания циклов методикой для нахождения параллелизма в программах с условными переходами, не связанными с организацией циклов. Трассировочное планирование полезно для машин с очень большим количеством команд, выдаваемых для выполнения в одном такте, где одного разворачивания циклов может оказаться недостаточно для выявления необходимой степени параллелизма уровня команд для поддержания машины в занятом состоянии. Трассировочное планирование является комбинацией двух отдельных процессов. Первый процесс, который называется выбором трассы (trace selection), старается найти возможную последовательность базовых блоков, операции которых будут собираться вместе в меньшее количество команд; эта последовательность называется трассой. Разворачивание циклов используется для генерации длинных трасс, поскольку переходы циклов выполняются с высокой вероятностью. Дополнительно при использовании статического прогнозирования направления переходов другие условные переходы (не связанные с организацией цикла) также выбираются как выполняемые или как невыполняемые, так что результирующая трасса представляет собой линейную последовательность, полученную в результате конкатенации (объединения) многих базовых блоков. Когда трасса выбрана, другой процесс, называемый уплотнением трассы (trace compaction), старается сжать трассу в небольшое количество широких команд. Процесс уплотнения трасс пытается перенести операции как можно ближе к началу последовательности (трассы), упаковывая операции насколько это возможно в минимальное количество широких команд (или пакетов для выдачи).

Уплотнение трассы представляет собой процесс глобального планирования кода. Имеются два разных ограничения, которые возникают и должны обрабатываться любой схемой глобальной оптимизации кода: зависимости по данным, которые задают определенный порядок операций, и точки условного перехода, которые создают места, через которые команды не могут просто перемещаться. По существу код должен быть уплотненным в наиболее короткую последовательность, которая сохраняет зависимости по данным и по управлению. Зависимости по данным преодолеваются посредством разворачивания циклов и использования анализа зависимостей для определения того, относятся ли два обращения к одному и тому же адресу. Зависимости по управлению также сокращаются при разворачивании циклов. Главным преимуществом методики трассировочного планирования по сравнению с более простым методом планирования загрузки конвейера заключается в том, что она обеспечивает схему для снижения эффекта зависимостей по управлению посредством переноса команд через условные переходы, не связанные с циклами, используя прогнозируемое поведение переходов. На рис. 6.17 показаны фрагменты кода, который может рассматриваться как итерация развернутого цикла, и выбранная трасса.

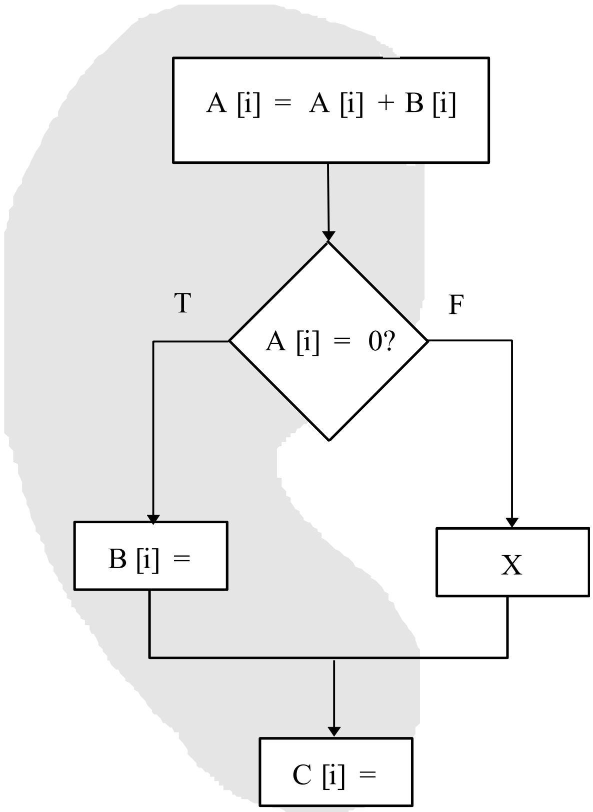


Рис. 6.17. Фрагмент кода с выбранной трассой

Когда трасса, как показано на рис. 6.17, выбрана, она должна быть уплотнена так, чтобы заполнить машинный ресурс. Уплотнение трассы приводит к перемещению операторов присваивания переменным B и C вверх по блоку, чтобы разместить их перед точкой решения о направлении перехода. Любая схема глобального планирования, включая трассировочное планирование, выполняет такое перемещение команд при наличии набора ограничений. В трассировочном планировании условные переходы рассматриваются как безусловные во внутрь или во вне выбранной трассы, которая предполагалась в качестве наиболее вероятный путь. Когда команды перемещаются через такие точки входа и выхода трассы, во входной и выходной точке могут потребоваться дополнительные команды. Главное предположение состоит в том, что выбранная трасса является наиболее вероятным событием, в противном случае стоимость дополнительной работы (дополнительных команд) может оказаться чрезмерной.

Что включает в себя процесс перемещения присваивания B и C? Вычисление и присваивание B является зависимым по управлению от условного перехода, а вычисление C нет. Перемещение этих операторов может быть выполнено только, если ни один из них не меняет зависимость по управлению или по данным, или эффект от изменения зависимости не виден и тем самым не приводит к изменению выполнения программы. Рассмотрим типичную последовательность генерации кода для диаграммы на рис. 6.16. Ниже представлена такая последовательность в предположении, что адреса для A, B и C находятся в регистрах R1, R2 и R3 соответственно:

```
            LW    R4,0(R1)       ;загрузка A[i]
            ADDI  R4,R4,...      ;добавление к A[i]
            SW    0(R1),R4       ;запись в A[i]
            ...
            BNEZ  R4,elsepart    ;проверка A[i]
            ...                  ;часть then
            SW    0(R2),...      ;запись в B[i]
            J     join           ;прыжок через else
elsepart:   ...                  ;часть else
            X                    ;код для X
            ...
join:       ..                   ;после if
            SW    0(R3),...      ;запись в C[i]
```

Сначала рассмотрим проблему перемещения операции присваивания B на место перед командой BNEZ. Поскольку B зависит по управлению от того перехода, перед которым мы ее хотим расположить, и не будет зависеть от него после перемещения, необходимо гарантировать, что выполнение оператора не может вызвать появление исключительной ситуации, поскольку такая исключительная ситуация не могла возникнуть в первоначальной программе, если бы была выбрана часть else условного оператора. Перемещение B не должно также воздействовать на поток данных. Чтобы более ясно определить требования, нам нужна концепция живучести переменной. Переменная Z живет в операторе, если имеется путь выполнения от этого оператора до использования переменной Z, на котором нового присваивания переменной Z не делается. На интуитивном уровне, переменная живет в операторе, если добавление операции присваивания этой переменной в операторе может изменить семантику программы.

Возвращаясь к нашему примеру, можно видеть, что имеются два возможных случая, когда перемещение B может изменить поток данных в этой программе:

1. 1. Обращение к B происходит в коде X (часть else) прежде, чем B будет присвоено новое значение.
2. 2. B "живет" в конце оператора if и ей не делается присваивания в X.

В обоих случаях перемещение операции присваивания переменной B приведет к тому, что некоторая команда i (либо в X, либо далее в программе) станет зависимой по данным от этой перемещенной команды, а не от более ранней операции присваивания B, которая выполняется перед циклом и от которой i первоначально зависела. Поскольку это приведет к изменению результата программы, операция присваивания B не может быть перемещена в случае, если справедливо любое из приведенных выше условий. Можно представить себе более изощренные схемы: например, в первом случае перед оператором if можно сделать теневую копию B и использовать только эту теневую копию в X. Такие схемы в общем случае не используются, поскольку, во-первых, их сложно реализовать, и, во-вторых, поскольку они будут замедлять программу, если выбранная трасса не оптимальна и завершение операций требует выполнения дополнительных команд.

Для перемещения операции присваивания C на место сразу за первым условным переходом требуется, чтобы она переместилась выше точки объединения трассы (входа трассы) с направлением else. Это делает команды для C зависимыми по управлению от

условного перехода и означает, что они не будут выполняться, если выбран путь else, который не находится на трассе. Поэтому будут затронуты команды, которые были зависимыми по данным от присваивания C и которые выполняются после этого кодового фрагмента. Для обеспечения вычисления корректного значения для этих команд делается копия команд, которые вычисляют и присваивают значение C на переходе на трасе, а именно в конце X на пути else. Мы можем переместить C из ветви then перехода через условие перехода, если это не воздействует ни на какой поток данных в условии перехода. Если C перемещается на место перед проверкой условия if, копия C в части else перехода может быть ликвидирована.
Все рассмотренные методы: разворачивание цикла, планирование трасс и программная конвейеризация стараются увеличить степень параллелизма уровня команд, который может использоваться машиной, выдающей для выполнения более одной команды в каждом такте. Эффективность каждого из этих методов и их удобство для различных архитектурных подходов являются наиболее горячими темами, которыми активно занимаются исследователи и разработчики высокоскоростных процессоров.

5.7. 6.7. Аппаратные средства поддержки большой степени распараллеливания

Методы, подобные разворачиванию циклов и планированию трасс, могут использоваться для увеличения степени доступного параллелизма, когда поведение условных переходов достаточно предсказуемо во время компиляции. Если же поведение переходов не известно, одной техники компиляторов может оказаться не достаточно для выявления большей степени параллелизма уровня команд. В этом разделе представлены два метода, которые могут помочь преодолеть подобные ограничения. Первый метод заключается в расширении набора команд условными или предикатными командами. Такие команды могут использоваться для ликвидации условных переходов и помогают компилятору перемещать команды через точки условных переходов. Условные команды увеличивают степень параллелизма уровня команд, но имеют существенные ограничения. Для использования большей степени параллелизма разработчики исследовали идею, которая называется "выполнением по предположению" (speculation), и позволяет выполнить команду еще до того, как процессор узнает, что она должна выполняться (т.е. этот метод позволяет избежать приостановок конвейера, связанных с зависимостями по управлению).
Условные команды
Концепция, лежащая в основе условных команд, достаточно проста: команда обращается к некоторому условию, оценка которого является частью выполнения команды. Если условие истинно, то команда выполняется нормально; если условие ложно, то выполнение команды осуществляется, как если бы это была пустая команда. Многие новейшие архитектуры включают в себя ту или иную форму условных команд. Наиболее общим примером такой команды является команда условной пересылки, которая выполняет пересылку значения одного регистра в другой, если условие истинно. Такая команда может использоваться для полного устранения условных переходов в простых последовательностях программного кода.
Например, рассмотрим следующий оператор:

```
if (A=0) {S=T;};
```

Предполагая, что регистры R1, R2 и R3 хранят значения A, S и T соответственно, представим код этого оператора с командой условного перехода и с командой условной пересылки.
Код с использованием команды условного перехода будет иметь следующий вид:

```
BEQZ  R1,L
```

```
        MOV    R2,R3
L:
```

Используя команду условной пересылки, которая выполняет пересылку, только если ее третий операнд равен нулю, мы можем реализовать этот оператор с помощью одной команды:

```
CMOVZ        R2,R3,R1
```

Условная команда позволяет преобразовать зависимость по управлению, присутствующую в коде с командой условного перехода, в зависимость по данным. (Это преобразование используется также в векторных машинах, в которых оно называется if-преобразованием (if-convertion)). Для конвейерной машины такое преобразование позволяет перенести точку, в которой должна разрешаться зависимость, от начала конвейера, где она разрешается для условных переходов, в конец конвейера, где происходит запись в регистр.

Одним из примеров использования команд условной пересылки является реализация функции вычисления абсолютного значения: A = abs (B), которая реализуется оператором

```
if (B<0) {A=-B} else {A=B}.
```

Этот оператор if может быть реализован парой команд условных пересылок или командой безусловной пересылки (A=B), за которой следует команда условной пересылки (A=-B).

Условные команды могут использоваться также для улучшения планирования в суперскалярных или VLIW-процессорах. Ниже приведен пример кодовой последовательности для суперскалярной машины с одновременной выдачей для выполнения не более двух команд. При этом в каждом такте может выдаваться комбинация одной команды обращения к памяти и одной команды АЛУ или только одна команда условного перехода:

```
LW      R1,40(R2)       ADD R3,R4,R5
                        ADD R6,R3,R7
BEQZ    R10,L
LW      R8,20(R10)
LW      R9,0(R8)
```

Эта последовательность теряет слот операции обращения к памяти во втором такте и приостанавливается из-за зависимости по данным, если переход невыполняемый, поскольку вторая команда LW после перехода зависит от предыдущей команды загрузки. Если доступна условная версия команды LW, то команда LW, немедленно следующая за переходом (LW R8,20(R10)), может быть перенесена во второй слот выдачи. Это улучшает время выполнения на несколько тактов, поскольку устраняет один слот выдачи команды и сокращает приостановку конвейера для последней команды последовательности.

Для успешного использования условных команд в примерах, подобных этому, семантика команды должна определять команду таким образом, чтобы не было никакого побочного эффекта, если условие не выполняется. Это означает, что если условие не выполняется, команда не должна записывать результат по месту назначения, а также не должна вызывать исключительную ситуацию. Как показывает вышеприведенный пример, способность не вызывать исключительную ситуацию достаточно важна: если регистр R10 содержит нуль, команда LW R8,20(R10), выполненная безусловно, возможно вызовет исключительную ситуацию по защите памяти, а эта исключительная ситуация не должна возникать. Именно эта вероятность возникновения исключительной ситуации не дает возможность компилятору просто перенести команду загрузки R8 через команду условного перехода. Конечно, если условие удовлетворено, команда LW все еще может вызвать исключительную ситуацию (например, ошибку страницы), и аппаратура должна воспринять эту исключительную ситуацию, поскольку она знает, что управляющее условие истинно.

Условные команды определенно полезны для реализации коротких альтернативных потоков управления. Тем не менее, полезность условных команд существенно ограничивается несколькими факторами:

- • Аннулируемые условные команды (т.е. команды, условие которых является ложным) все же отнимают определенное время выполнения. Поэтому перенос команды через команду условного перехода и превращение ее в условную будет замедлять программу всякий раз, когда перенесенная команда не будет нормально выполняться. Важное исключение из этого правила возникает, когда такты, используемые перенесенной невыполняемой командой, были бы в любом случае холостыми (как в вышеприведенном примере с суперскалярной обработкой). Перенос команды через команду условного перехода существенно базируется на предположении о направлении перехода. Условные команды упрощают реализацию такого переноса, но не устраняют время выполнения, которое будет затрачено при неправильном предположении.
- • Условные команды наиболее полезны, когда условие может быть вычислено заранее. Если условие и условный переход не могут быть отделены друг от друга (из-за зависимости по данным при определении условия), то условная команда не поможет, хотя все еще может оказаться полезной, поскольку она задерживает момент времени, когда условие должно стать известным, почти до конца конвейера.
- • Использование условных команд ограничено, когда в поток управления вовлечено больше одной простой альтернативной последовательности команд. Например, при переносе команды через пару команд условного перехода необходимо, чтобы она оставалась зависимой от обоих условий, что требует либо спецификации в команде сразу двух условий (маловероятная возможность), либо вставки дополнительных команд для вычисления конъюнкции условий.
- • Условные команды могут давать некоторые потери скорости по сравнению с безусловными командами. Это может проявиться либо в большем количестве тактов, необходимых для выполнения таких команд, либо в уменьшении общей частоты синхронизации машины. Если условные команды являются более дорогими с точки зрения скорости выполнения, то их следует использовать осмысленно.

По этим причинам во многих современных архитектурах используется небольшое число условных команд (наиболее популярными являются команды условных пересылок), хотя некоторые из них включают условные версии большинства команд (рис. 6.18).

Alpha	HP-PA	MIPS	PowerPC	SPARC
Условная пересылка	Любая команда типа регистр-регистр может аннулировать следующую команду, делая ее условной	Условная пересылка	Условная пересылка	Условная пересылка

Рис. 6.18. Условные команды в современных архитектурах

Выполнение по предположению (speculation)

Поддерживаемое аппаратурой выполнение по предположению позволяет выполнить команду до момента определения направления условного перехода, от которого данная команда зависит. Это снижает потери, которые возникают при наличии в программе зависимостей по управлению. Чтобы понять, почему выполнение по предположению оказывается полезным, рассмотрим следующий простой пример программного кода, который реализует проход по связанному списку и инкрементирование каждого элемента этого списка:

```
for (p=head; p <> nil; *p=*p.next) {
        *p.value = *p.value+1;
    }
```

Подобно циклам for, с которыми мы встречались в более ранних разделах, разворачивание этого цикла не увеличит степени доступного параллелизма уровня команд. Действительно, каждая развернутая итерация будет содержать оператор if и выход из цикла. Ниже приведена последовательность команд в предположении, что значение head находится в регистре R4, который используется для хранения p, и что каждый элемент списка состоит из поля значения и следующего за ним поля указателя. Проверка размещается внизу так, что на каждой итерации цикла выполняется только один переход.

```
                J       looptest
start:          LW      R5,0(R4)
                ADDI    R5,R5,#1
                SW      0(R4),R5
                LW      R4,4(R4)
looptest:       BNEZ    R4,start
```

Развернув цикл однажды можно видеть, что разворачивание в данном случае не помогает:

```
                J       looptest
start:          LW      R5,0(R4)
                ADDI    R5,R5,#1
                SW      0(R4),R5
                LW      R4,4(R4)
                BNEZ    R4,end
                LW      R5,0(R4)
                ADDI    R5,R5,#1
                SW      0(R4),R5
                LW      R4,4(R4)
looptest:       BNEZ    R4,start
end:
```

Даже прогнозируя направление перехода, мы не можем выполнять с перекрытием команды из двух разных итераций цикла, и условные команды в любом случае здесь не помогут. Имеются несколько сложных моментов для выявления параллелизма из этого развернутого цикла:

- • Первая команда в итерации цикла (LW R5,0(R4)) зависит по управлению от обоих условных переходов. Таким образом, команда не может выполняться успешно (и безопасно) до тех пор, пока мы не узнаем исходы команд перехода.
- • Вторая и третья команды в итерации цикла зависят по данным от первой команды цикла.
- • Четвертая команда в каждой итерации цикла (LW R4,4(R4)) зависит по управлению от обоих переходов и антизависит от непосредственно предшествующей ей команды SW.

- • Последняя команда итерации цикла зависит от четвертой. Вместе эти условия означают, что мы не можем совмещать выполнение никаких команд между последовательными итерациями цикла! Имеется небольшая возможность совмещения посредством переименования регистров либо аппаратными, либо программными средствами, если цикл развернут, так что вторая загрузка более не антизависит от SW и может быть перенесена выше.

В альтернативном варианте, при выполнении по предположению, что переход не будет выполняться, мы можем попытаться совместить выполнение последовательных итераций цикла. Действительно, это в точности то, что делает компилятор с планированием трасс. Когда направление переходов может прогнозироваться во время компиляции, и компилятор может найти команды, которые он может безопасно перенести на место перед точкой перехода, решение, базирующееся на технологии компилятора, идеально. Эти два условия являются ключевыми ограничениями для выявления параллелизма уровня команд статически с помощью компилятора. Рассмотрим развернутый выше цикл. Переход просто трудно прогнозируем, поскольку частота, с которой он является выполняемым, зависит от длины списка, по которому осуществляется проход. Кроме того, мы не можем безопасно перенести команду загрузки через переход, поскольку, если содержимое R4 равно nil, то команда загрузки слова, которая использует R4 как базовый регистр, гарантированно приведет к ошибке и обычно сгенерирует исключительную ситуацию по защите. Во многих системах значение nil реализуется с помощью указателя на неиспользуемую страницу виртуальной памяти, что обеспечивает ловушку (trap) при обращении по нему. Такое решение хорошо для универсальной схемы обнаружения указателей на nil, но в данном случае это не очень помогает, поскольку мы можем регулярно генерировать эту исключительную ситуацию, и стоимость обработки исключительной ситуации плюс уничтожения результатов выполнения по предположению будет огромной.

Чтобы преодолеть эти сложности, машина может иметь в своем составе специальные аппаратные средства поддержки выполнения по предположению. Эта методика позволяет машине выполнять команду, которая может быть зависимой по управлению, и избежать любых последствий выполнения этой команды (включая исключительные ситуации), если окажется, что в действительности команда не должна выполняться. Таким образом выполнение по предположению, подобно условным командам, позволяет преодолеть два сложных момента, которые могут возникнуть при более раннем выполнении команд: возможность появления исключительной ситуации и ненужное изменение состояния машины, вызванное выполнением команды. Кроме того, механизмы выполнения по предположению позволяют выполнять команду даже до момента оценки условия командой условного перехода, что невозможно при условных командах. Конечно, аппаратная поддержка выполнения по предположению достаточно сложна и требует значительных аппаратных ресурсов.

Один из подходов, который был хорошо исследован во множестве исследовательских проектов и используется в той или иной степени в машинах, которые разработаны или находятся на стадии разработки в настоящее время, заключается в объединении аппаратных средств динамического планирования и выполнения по предположению. В определенной степени подобную работу делала и IBM 360/91, поскольку она могла использовать средства прогнозирования направления переходов для выборки команд и назначения этих команд на станции резервирования. Механизмы, допускающие выполнение по предположению, идут дальше и позволяют действительно выполнять эти команды, а также другие команды, зависящие от команд, выполняющихся по предположению. Как и для алгоритма Томасуло, поясним аппаратное выполнение по предположению на примере устройства плавающей точки, но все идеи естественно применимы и для целочисленного устройства.

Аппаратура, реализующая алгоритм Томасуло, может быть расширена для обеспечения поддержки выполнения по предположению. С этой целью необходимо отделить средства пересылки результатов команд, которые требуются для выполнения по предположению некоторой команды, от механизма действительного завершения команды. Имея такое разделение функций, мы можем допустить выполнение команды и пересылать ее результаты другим командам, не позволяя ей однако делать никакие обновления состояния машины, которые не могут быть ликвидированы, до тех пор, пока мы не узнаем, что команда должна безусловно выполниться. Использование цепей ускоренной пересылки также подобно выполнению по предположению чтения регистра, поскольку мы не знаем, обеспечивает ли команда, формирующая значение регистра-источника, корректный результат до тех пор, пока ее выполнение не станет безусловным. Если команда, выполняемая по предположению, становится безусловной, ей разрешается обновить регистровый файл или память. Этот дополнительный этап выполнения команд обычно называется стадией фиксации результатов команды (instruction commit).

Главная идея, лежащая в основе реализации выполнения по предположению, заключается в разрешении неупорядоченного выполнения команд, но в строгом соблюдении порядка фиксации результатов и предотвращением любого безвозвратного действия (например, обновления состояния или приема исключительной ситуации) до тех пор, пока результат команды не фиксируется. В простом конвейере с выдачей одиночных команд мы могли бы гарантировать, что команда фиксируется в порядке, предписанном программой, и только после проверки отсутствия исключительной ситуации, вырабатываемой этой командой, просто посредством переноса этапа записи результата в конец конвейера. Когда мы добавляем механизм выполнения по предположению, мы должны отделить процесс фиксации команды, поскольку он может произойти намного позже, чем в простом конвейере. Добавление к последовательности выполнения команды этой фазы фиксации требует некоторых изменений в последовательности действий, а также дополнительного набора аппаратных буферов, которые хранят результаты команд, которые завершили выполнение, но еще не зафиксированы. Этот аппаратный буфер, который можно назвать буфером переупорядочивания, используется также для передачи результатов между командами, которые могут выполняться по предположению.

Буфер переупорядочивания предоставляет дополнительные виртуальные регистры точно так же, как станции резервирования в алгоритме Томасуло расширяют набор регистров. Буфер переупорядочивания хранит результат некоторой операции в промежутке времени от момента завершения операции, связанной с этой командой, до момента фиксации результатов команды. Поэтому буфер переупорядочивания является источником операндов для команд, точно также как станции резервирования обеспечивают промежуточное хранение и передачу операндов в алгоритме Томасуло. Основная разница заключается в том, что когда в алгоритме Томасуло команда записывает свой результат, любая последующая выдаваемая команда будет выбирать этот результат из регистрового файла. При выполнении по предположению регистровый файл не обновляется до тех пор, пока команда не фиксируется (и мы знаем определенно, что команда должна выполняться); таким образом, буфер переупорядочивания поставляет операнды в интервале между завершением выполнения и фиксацией результатов команды. Буфер переупорядочивания не похож на буфер записи в алгоритме Томасуло, и в нашем примере функции буфера записи интегрированы с буфером переупорядочивания только с целью упрощения. Поскольку буфер переупорядочивания отвечает за хранение результатов до момента их записи в регистры, он также выполняет функции буфера загрузки.

Каждая строка в буфере переупорядочивания содержит три поля: поле типа команды, поле места назначения (результата) и поле значения. Поле типа команды определяет, является ли команда условным переходом (для которого отсутствует место назначения результата), командой записи (которая в качестве места назначения результата использует адрес памяти) или регистровой операцией (команда АЛУ или команда

загрузки, в которых местом назначения результата является регистр). Поле назначения обеспечивает хранение номера регистра (для команд загрузки и АЛУ) или адрес памяти (для команд записи), в который должен быть записан результат команды. Поле значения используется для хранения результата операции до момента фиксации результата команды. На рис. 6.19 показана аппаратная структура машины с буфером переупорядочивания. Буфер переупорядочивания полностью заменяет буфера загрузки и записи. Хотя функция переименования станций резервирования заменена буфером переупорядочивания, нам все еще необходимо некоторое место для буферизации операций (и операндов) между моментом их выдачи и началом выполнения. Эту функцию выполняют регистровые станции резервирования. Поскольку каждая команда имеет позицию в буфере переупорядочивания до тех пор, пока она не будет зафиксирована (и результаты не будут отправлены в регистровый файл), результат тегируется посредством номера строки буфера переупорядочивания, а не номером станции резервирования. Это требует, чтобы номер строки буфера переупорядочивания, присвоенный команде, отслеживался станцией резервирования.
Ниже перечислены четыре этапа выполнение команды:

1. 1. *Выдача.* Получает команду из очереди команд плавающей точки. Выдает команду для выполнения, если имеется свободная станция резервирования и свободная строка в буфере переупорядочивания; передает на станцию резервирования операнды, если они находятся в регистрах или в буфере переупорядочивания; и обновляет поля управления для индикации того, что буфера используются. Номер отведенной под результат строки буфера переупорядочивания также записывается в станцию резервирования, так что этот номер может использоваться для тегирования (пометки) результата, когда он помещается на CDB. Если все станции резервирования заполнены, или полон буфер переупорядочивания, выдача команды приостанавливается до тех пор, пока в обоих буферах не появится доступной строки.
2. 2. *Выполнение.* Если один или несколько операндов еще не готовы (отсутствуют), осуществляется просмотр CDB (Common Data Bus) и происходит ожидание вычисления значения требуемого регистра. На этом шаге выполняется проверка наличия конфликтов типа RAW. Когда оба операнда оказываются на станции резервирования, происходит вычисление результата операции.
3. 3. *Запись результата.* Когда результат вычислен и становится доступным, выполняется его запись на CDB (с посылкой тега буфера переупорядочивания, который был присвоен команде на этапе выдачи для выполнения) и из CDB в буфер переупорядочивания, а также в каждую станцию резервирования, ожидающую этот результат. (Можно было бы также читать результат из буфера переупорядочивания, а не из CDB, точно так же, как централизованная схема управления (scoreboard) читает результаты из регистров, а не с шины завершения). Станция резервирования помечается как свободная.
4. 4. *Фиксация.* Когда команда достигает головы буфера переупорядочивания и ее результат присутствует в буфере, соответствующий регистр обновляется значением результата (или выполняется запись в память, если операция - запись в память), и команда изымается из буфера переупорядочивания.

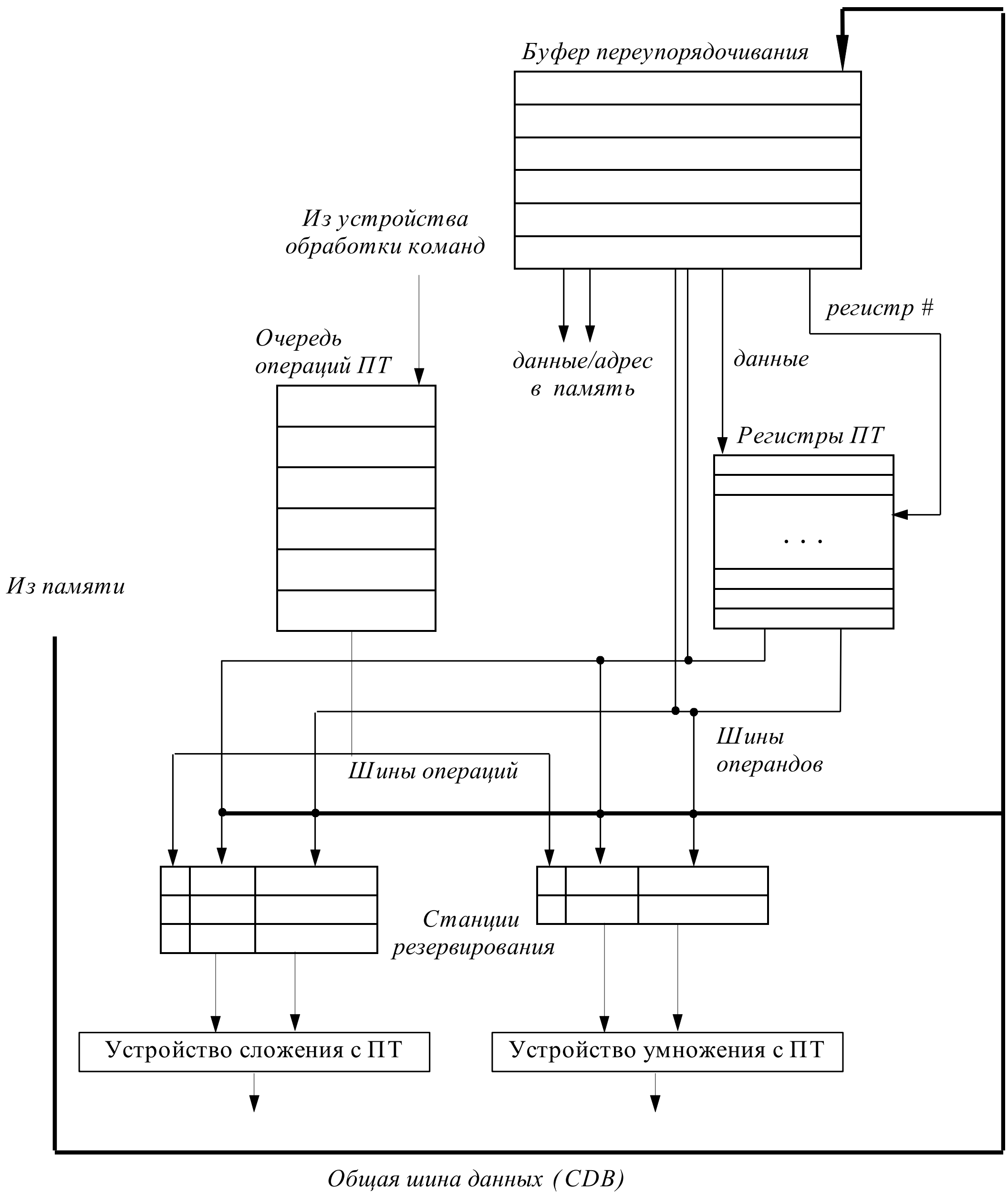


Рис. 6.19. Расширение устройства ПТ средствами выполнения по предположению

Когда команда фиксируется, соответствующая строка буфера переупорядочивания очищается, а место назначения результата (регистр или ячейка памяти) обновляется. Чтобы не менять номера строк буфера переупорядочивания после фиксации результата команды, буфер переупорядочивания реализуется в виде циклической очереди, так что позиции в буфере переупорядочивания меняются, только когда команда фиксируется. Если буфер переупорядочивания полностью заполнен, выдача команд останавливается до тех пор, пока не освободится очередная строка буфера.

Поскольку никакая запись в регистры или ячейки памяти не происходит до тех пор, пока команда не фиксируется, машина может просто ликвидировать все свои выполненные по предположению действия, если обнаруживается, что направление условного перехода было спрогнозировано не верно.

Рассмотрим следующий пример:

LD F6, 34(R2)
LD F2, 45(R3)
MULTD F0, F2, F4
SUBD F8, F6, F2
DIVD F10, F0, F6
ADDD F6, F8, F2

Представим, что в приведенном выше примере команда условного перехода BNEZ в первый раз не выполняется (рис. 6.20). Тогда команды, предшествующие команде условного перехода, будут просто фиксироваться по мере достижения каждой из них головы буфера переупорядочивания. Когда головы этого буфера достигает команда условного перехода, содержимое буфера просто гасится, и машина начинает выборку команд из другой ветви программы.

Исключительные ситуации в подобной машине не воспринимаются до тех пор, пока соответствующая команда не готова к фиксации. Если выполняемая по предположению команда вызывает исключительную ситуацию, эта исключительная ситуация записывается в буфер упорядочивания. Если обнаруживается неправильный прогноз направления условного перехода и выясняется, что команда не должна была выполняться, исключительная ситуация гасится вместе с командой, когда обнуляется буфер переупорядочивания. Если же команда достигает вершины буфера переупорядочивания, то мы знаем, что она более не является выполняемой по предположению (она уже стала безусловной), и исключительная ситуация должна действительно восприниматься.

Эту методику выполнения по предположению легко распространить и на целочисленные регистры и функциональные устройства. Действительно, выполнение по предположению может быть более полезно в целочисленных программах, поскольку именно такие программы имеют менее предсказуемое поведение переходов. Кроме того, эти методы могут быть расширены так, чтобы обеспечить работу в машинах с выдачей на выполнение и фиксацией результатов нескольких команд в каждом такте. Выполнение по предположению возможно является наиболее интересным методом именно для таких машин, поскольку менее амбициозные машины могут довольствоваться параллелизмом уровня команд внутри базовых блоков при соответствующей поддержке со стороны компилятора, использующего технологию разворачивания циклов.

Очевидно, все рассмотренные ранее методы не могут достичь большей степени распараллеливания, чем заложено в конкретной прикладной программе. Вопрос увеличения степени параллелизма прикладных систем в настоящее время является предметом интенсивных исследований, проводимых во всем мире.

Станции резервирования							
Имя	**Занятость**	**Op**	**V_j**	**V_k**	**Q_j**	**Q_k**	**Назначение**
Add1	Нет						
Add2	Нет						
Add3	Нет						
Mult1	Нет	MULT	Mem[45+Regs[R3]]	Regs[F4]			#3

Mult2	Да	DIV	Mem[34+Regs[R2]]	#3	#5

Буфер переупорядочивания					
Номер	**Занятость**	**Команда**	**Состояние**	**Место назначения**	**Значение**
1	Нет	`LD F6,34(R2)`	Зафиксирована	F6	Mem[34+Regs[R2]]
2	Нет	`LD F2,45(R3)`	Зафиксирована	F2	Mem[45+Regs[R3]]
3	Нет	`MULTD F0,F2,F4`	Запись результата	F0	#2 x Regs[R4]]
4	Да	`SUBD F8,F6,F2`	Запись результата	F8	#1 - #2
5	Да	`DIVD F10,F0,F6`	Выполнение	F10	
6	Да	`ADD F6,F8,F2`	Запись результата	F6	#4 + #2

Состояние регистров									
Поле	**F0**	**F2**	**F4**	**F6**	**F8**	**F10**	**F12**	**. . .**	**F30**
Порядковый #	3			6	4	5			
Знятость	Да	Нет	Нет	Да	Да	Да	Нет	. . .	Нет

Рис. 6.20. Состояние устройства ПТ для выполнения по предположению

6. 7. Иерархия памяти

6.1. 7.1. Введение

В основе реализации иерархии памяти современных компьютеров лежат два принципа: принцип локальности обращений и соотношение стоимость/производительность. Принцип локальности обращений говорит о том, что большинство программ к счастью не выполняют обращений ко всем своим командам и данным равновероятно, а оказывают предпочтение некоторой части своего адресного пространства.

Иерархия памяти современных компьютеров строится на нескольких уровнях, причем более высокий уровень меньше по объему, быстрее и имеет большую стоимость в пересчете на байт, чем более низкий уровень. Уровни иерархии взаимосвязаны: все данные на одном уровне могут быть также найдены на более низком уровне, и все данные на этом более низком уровне могут быть найдены на следующем нижележащем уровне и так далее, пока мы не достигнем основания иерархии.

Иерархия памяти обычно состоит из многих уровней, но в каждый момент времени мы имеем дело только с двумя близлежащими уровнями. Минимальная единица информации, которая может либо присутствовать, либо отсутствовать в двухуровневой иерархии, называется блоком. Размер блока может быть либо фиксированным, либо переменным. Если этот размер зафиксирован, то объем памяти является кратным размеру блока.

Успешное или неуспешное обращение к более высокому уровню называются соответственно попаданием (hit) или промахом (miss). Попадание - есть обращение к объекту в памяти, который найден на более высоком уровне, в то время как промах означает, что он не найден на этом уровне. Доля попаданий (hit rate) или коэффициент попаданий (hit ratio) есть доля обращений, найденных на более высоком уровне. Иногда она представляется процентами. Доля промахов (miss rate) есть доля обращений, которые не найдены на более высоком уровне.

Поскольку повышение производительности является главной причиной появления иерархии памяти, частота попаданий и промахов является важной характеристикой. Время обращения при попадании (hit time) есть время обращения к более высокому уровню иерархии, которое включает в себя, в частности, и время, необходимое для определения того, является ли обращение попаданием или промахом. Потери на промах (miss penalty) есть время для замещения блока в более высоком уровне на блок из более низкого уровня плюс время для пересылки этого блока в требуемое устройство (обычно в процессор). Потери на промах далее включают в себя две компоненты: время доступа (access time) - время обращения к первому слову блока при промахе, и время пересылки (transfer time) - дополнительное время для пересылки оставшихся слов блока. Время доступа связано с задержкой памяти более низкого уровня, в то время как время пересылки связано с полосой пропускания канала между устройствами памяти двух смежных уровней.

Чтобы описать некоторый уровень иерархии памяти надо ответить на следующие четыре вопроса:

1. Где может размещаться блок на верхнем уровне иерархии? (размещение блока).
2. Как найти блок, когда он находится на верхнем уровне? (идентификация блока).
3. Какой блок должен быть замещен в случае промаха? (замещение блоков).
4. Что происходит во время записи? (стратегия записи).

6.2. **7.2. Организация кэш-памяти**

Концепция кэш-памяти возникла раньше, чем архитектура IBM/360, и сегодня кэш-память имеется практически в любом классе компьютеров, а в некоторых компьютерах - во множественном числе.
Все термины, которые были определены раньше, могут быть использованы и для кэш-памяти, хотя слово "строка" (line) часто употребляется вместо слова "блок" (block).
На рис. 7.1 представлен типичный набор параметров, который используется для описания кэш-памяти.

Размер блока (строки)	4-128 байт
Время попадания (hit time)	1-4 такта синхронизации (обычно 1 такт)
Потери при промахе (miss penalty) (Время доступа - access time) (Время пересылки - transfer time)	8-32 такта синхронизации (6-10 тактов синхронизации) (2-22 такта синхронизации)
Доля промахов (miss rate)	1%-20%
Размер кэш-памяти	4 Кбайт - 16 Мбайт

Рис. 7.1. Типовые значения ключевых параметров для кэш-памяти рабочих станций и серверов.

Рассмотрим организацию кэш-памяти более детально, отвечая на четыре вопроса об иерархии памяти.

1. Где может размещаться блок в кэш-памяти?
Принципы размещения блоков в кэш-памяти определяют три основных типа их организации:

- • Если каждый блок основной памяти имеет только одно фиксированное место, на котором он может появиться в кэш-памяти, то такая кэш-память называется кэшем с прямым отображением (direct mapped). Это наиболее простая организация кэш-памяти, при которой для отображения адресов блоков основной памяти на адреса кэш-памяти просто используются младшие разряды адреса блока. Таким образом, все блоки основной памяти, имеющие одинаковые младшие разряды в своем адресе, попадают в один блок кэш-памяти, т.е.
(адрес блока кэш-памяти) =
(адрес блока основной памяти) mod (число блоков в кэш-памяти)
- • Если некоторый блок основной памяти может располагаться на любом месте кэш-памяти, то кэш называется полностью ассоциативным (fully associative).
- • Если некоторый блок основной памяти может располагаться на ограниченном множестве мест в кэш-памяти, то кэш называется множественно-ассоциативным (set associative). Обычно множество представляет собой группу из двух или большего числа блоков в кэше. Если множество состоит из n блоков, то такое размещение называется множественно-ассоциативным с n каналами (n-way set associative). Для размещения блока, прежде всего, необходимо определить множество. Множество определяется младшими разрядами адреса блока памяти (индексом):

(адрес множества кэш-памяти) =
(адрес блока основной памяти) mod (число множеств в кэш-памяти)

Далее, блок может размещаться на любом месте данного множества.

Диапазон возможных организаций кэш-памяти очень широк: кэш-память с прямым отображением есть просто одноканальная множественно-ассоциативная кэш-память, а полностью ассоциативная кэш-память с m блоками может быть названа m-канальной множественно-ассоциативной. В современных процессорах, как правило, используется либо кэш-память с прямым отображением, либо двух- (четырех-) канальная множественно-ассоциативная кэш-память.

2. Как найти блок, находящийся в кэш-памяти?

У каждого блока в кэш-памяти имеется адресный тег, указывающий, какой блок в основной памяти данный блок кэш-памяти представляет. Эти теги обычно одновременно сравниваются с выработанным процессором адресом блока памяти.

Кроме того, необходим способ определения того, что блок кэш-памяти содержит достоверную или пригодную для использования информацию. Наиболее общим способом решения этой проблемы является добавление к тегу так называемого бита достоверности (valid bit).

Адресация множественно-ассоциативной кэш-памяти осуществляется путем деления адреса, поступающего из процессора, на три части: поле смещения используется для выбора байта внутри блока кэш-памяти, поле индекса определяет номер множества, а поле тега используется для сравнения. Если общий размер кэш-памяти зафиксировать, то увеличение степени ассоциативности приводит к увеличению количества блоков в множестве, при этом уменьшается размер индекса и увеличивается размер тега.

3. Какой блок кэш-памяти должен быть замещен при промахе?

При возникновении промаха, контроллер кэш-памяти должен выбрать подлежащий замещению блок. Польза от использования организации с прямым отображением заключается в том, что аппаратные решения здесь наиболее простые. Выбирать просто нечего: на попадание проверяется только один блок и только этот блок может быть замещен. При полностью ассоциативной или множественно-ассоциативной организации кэш-памяти имеются несколько блоков, из которых надо выбрать кандидата в случае промаха. Как правило для замещения блоков применяются две основных стратегии: случайная и LRU.

В первом случае, чтобы иметь равномерное распределение, блоки-кандидаты выбираются случайно. В некоторых системах, чтобы получить воспроизводимое поведение, которое особенно полезно во время отладки аппаратуры, используют псевдослучайный алгоритм замещения.

Во втором случае, чтобы уменьшить вероятность выбрасывания информации, которая скоро может потребоваться, все обращения к блокам фиксируются. Заменяется тот блок, который не использовался дольше всех (LRU - Least-Recently Used).

Достоинство случайного способа заключается в том, что его проще реализовать в аппаратуре. Когда количество блоков для поддержания трассы увеличивается, алгоритм LRU становится все более дорогим и часто только приближенным. На рис. 7.2 показаны различия в долях промахов при использовании алгоритма замещения LRU и случайного алгоритма.

Ассоциативность:	2-канальная	4-канальная	8-канальная
Размер кэш-памяти	LRU Random	LRU Random	LRU Random
16 KB	5.18% 5.69%	4.67% 5.29%	4.39% 4.96%
64 KB	1.88% 2.01%	1.54% 1.66%	1.39% 1.53%
256 KB	1.15% 1.17%	1.13% 1.13%	1.12% 1.12%

Рис. 7.2. Сравнение долей промахов для алгоритма LRU и случайного алгоритма замещения при нескольких размерах кэша и разных ассоциативностях при размере блока 16 байт.

4. Что происходит во время записи?

При обращениях к кэш-памяти на реальных программах преобладают обращения по чтению. Все обращения за командами являются обращениями по чтению и большинство команд не пишут в память. Обычно операции записи составляют менее 10% общего трафика памяти. Желание сделать общий случай более быстрым означает оптимизацию кэш-памяти для выполнения операций чтения, однако при реализации высокопроизводительной обработки данных нельзя пренебрегать и скоростью операций записи.

К счастью, общий случай является и более простым. Блок из кэш-памяти может быть прочитан в то же самое время, когда читается и сравнивается его тег. Таким образом, чтение блока начинается сразу как только становится доступным адрес блока. Если чтение происходит с попаданием, то блок немедленно направляется в процессор. Если же происходит промах, то от заранее считанного блока нет никакой пользы, правда нет и никакого вреда.

Однако при выполнении операции записи ситуация коренным образом меняется. Именно процессор определяет размер записи (обычно от 1 до 8 байтов) и только эта часть блока может быть изменена. В общем случае это подразумевает выполнение над блоком последовательности операций чтение-модификация-запись: чтение оригинала блока, модификацию его части и запись нового значения блока. Более того, модификация блока не может начинаться до тех пор, пока проверяется тег, чтобы убедиться в том, что обращение является попаданием. Поскольку проверка тегов не может выполняться параллельно с другой работой, то операции записи отнимают больше времени, чем операции чтения.

Очень часто организация кэш-памяти в разных машинах отличается именно стратегией выполнения записи. Когда выполняется запись в кэш-память, имеются две базовые возможности:

- • сквозная запись (write through, store through) - информация записывается в два места: в блок кэш-памяти и в блок более низкого уровня памяти.
- • запись с отложенным обратным копированием (write back, copy back, store in) - информация записывается только в блок кэш-памяти. Модифицированный блок кэш-памяти записывается в основную память только тогда, когда он замещается. Для сокращения частоты копирования блоков при замещении обычно с каждым блоком кэш-памяти связывается так называемый бит модификации (dirty bit). Этот бит состояния показывает, был ли модифицирован блок, находящийся в кэш-памяти. Если он не модифицировался, то обратное копирование отменяется, поскольку более низкий уровень содержит ту же самую информацию, что и кэш-память.

Оба подхода к организации записи имеют свои преимущества и недостатки. При записи с обратным копированием операции записи выполняются со скоростью кэш-памяти, и несколько записей в один и тот же блок требуют только одной записи в память более низкого уровня. Поскольку в этом случае обращения к основной памяти происходят реже, вообще говоря требуется меньшая полоса пропускания памяти, что очень привлекательно для мультипроцессорных систем. При сквозной записи промахи по чтению не влияют на записи в более высокий уровень, и, кроме того, сквозная запись проще для реализации, чем запись с обратным копированием. Сквозная запись имеет также преимущество в том, что основная память имеет наиболее свежую копию данных. Это важно в мультипроцессорных системах, а также для организации ввода/вывода.

Когда процессор ожидает завершения записи при выполнении сквозной записи, то говорят, что он приостанавливается для записи (write stall). Общий прием минимизации остановов по записи связан с использованием буфера записи (write buffer), который позволяет процессору продолжить выполнение команд во время обновления содержимого памяти. Следует отметить, что остановы по записи могут возникать и при наличии буфера записи.

При промахе во время записи имеются две дополнительные возможности:

- • поместить запись в кэш-память (write allocate) (называется также выборкой при записи (fetch on write)). Блок загружается в кэш-память, после чего выполняются действия, аналогичные выполняющимся при выполнении записи с попаданием. Это похоже на промах при чтении.
- • не размещать запись в кэш-памяти (называется также записью в окружение (write around)). Блок модифицируется на более низком уровне и не загружается в кэш-память.

Обычно в кэш-памяти, реализующей запись с обратным копированием, используется размещение записи в кэш-памяти (в надежде, что последующая запись в этот блок будет перехвачена), а в кэш-памяти со сквозной записью размещение записи в кэш-памяти часто не используется (поскольку последующая запись в этот блок все равно пойдет в память).

Увеличение производительности кэш-памяти

Формула для среднего времени доступа к памяти в системах с кэш-памятью выглядит следующим образом:

Среднее время доступа = Время обращения при попадании + Доля промахов x Потери при промахе

Эта формула наглядно показывает пути оптимизации работы кэш-памяти: сокращение доли промахов, сокращение потерь при промахе, а также сокращение времени обращения к кэш-памяти при попадании. Ниже на рис. 7.3 кратко представлены различные методы, которые используются в настоящее время для увеличения производительности кэш-памяти. Использование тех или иных методов определяется прежде всего целью разработки, при этом конструкторы современных компьютеров заботятся о том, чтобы система оказалась сбалансированной по всем параметрам.

Метод	Доля промахов	Потери при промахе	Время обраще-ния при попадании	Слож-ность аппаратуры	Примечания
Увеличение размера блока	+	-		0	
Повышение степени ассоциативности	+		-	1	
Кэш-память с *вспомогательным кэшем*	+			2	
Псевдоассоциативные кэши	+			2	
Аппаратная предварительная выборка команд и данных	+			2	Предварительная выборка данных затруднена
Предварительная выборка под управлением компилятора	+			3	Требует также неблокируемой кэш-памяти
Специальные методы для уменьшения промахов	+			0	Вопрос ПО
Установка приоритетов промахов по чтению над записями		+		1	Просто для однопроцессорных систем
Использование подблоков		+	+	1	Сквозная запись + подблок на 1 слово помогают записям
Пересылка требуемого слова первым		+		2	
Неблокируемые кэши		+		3	

Кэши второго уровня		+		2	Достаточно дорогое оборудование
Простые кэши малого размера	-		+	0	
Обход преобразования адресов во время индексации кэш-памяти			+	2	
Конвейеризация операций записи для быстрого попадания при записи			+	1	

Рис. 7.3. Обобщение методов оптимизации кэш-памяти

6.3. **7.3. Принципы организации основной памяти в современных компьютерах**

6.3.1. **7.3.1. Общие положения**

Основная память представляет собой следующий уровень иерархии памяти. Основная память удовлетворяет запросы кэш-памяти и служит в качестве интерфейса ввода/вывода, поскольку является местом назначения для ввода и источником для вывода. Для оценки производительности основной памяти используются два основных параметра: задержка и полоса пропускания. Традиционно задержка основной памяти имеет отношение к кэш-памяти, а полоса пропускания или пропускная способность относится к вводу/выводу. В связи с ростом популярности кэш-памяти второго уровня и увеличением размеров блоков у такой кэш-памяти, полоса пропускания основной памяти становится важной также и для кэш-памяти.

Задержка памяти традиционно оценивается двумя параметрами: временем доступа (access time) и длительностью цикла памяти (cycle time). Время доступа представляет собой промежуток времени между выдачей запроса на чтение и моментом поступления запрошенного слова из памяти. Длительность цикла памяти определяется минимальным временем между двумя последовательными обращениями к памяти.

Основная память современных компьютеров реализуется на микросхемах статических и динамических ЗУПВ (Запоминающее Устройство с Произвольной Выборкой). Микросхемы статических ЗУВП (СЗУПВ) имеют меньшее время доступа и не требуют циклов регенерации. Микросхемы динамических ЗУПВ (ДЗУПВ) характеризуются большей емкостью и меньшей стоимостью, но требуют схем регенерации и имеют значительно большее время доступа.

В процессе развития ДЗУВП с ростом их емкости основным вопросом стоимости таких микросхем был вопрос о количестве адресных линий и стоимости соответствующего корпуса. В те годы было принято решение о необходимости мультиплексирования адресных линий, позволившее сократить наполовину количество контактов корпуса, необходимых для передачи адреса. Поэтому обращение к ДЗУВП обычно происходит в два этапа. Первый этап начинается с выдачи сигнала RAS - row-access strobe (строб адреса строки), который фиксирует в микросхеме поступивший адрес строки. Второй этап включает переключение адреса для указания адреса столбца и подачу сигнала CAS - column-access stobe (строб адреса столбца), который фиксирует этот адрес и разрешает работу выходных буферов микросхемы. Названия этих сигналов связаны с внутренней организацией микросхемы, которая, как правило, представляет собой прямоугольную матрицу, к элементам которой можно адресоваться с помощью указания адреса строки и адреса столбца.

Дополнительным требованием организации ДЗУВП является необходимость периодической регенерации ее состояния. При этом все биты в строке могут регенерироваться одновременно, например, путем чтения этой строки. Поэтому ко

всем строкам всех микросхем ДЗУПВ основной памяти компьютера должны производиться периодические обращения в пределах определенного временного интервала порядка 8 миллисекунд.
Это требование кроме всего прочего означает, что система основной памяти компьютера оказывается иногда недоступной процессору, так как она вынуждена рассылать сигналы регенерации каждой микросхеме. Разработчики ДЗУПВ стараются поддерживать время, затрачиваемое на регенерацию, на уровне менее 5% общего времени. Обычно контроллеры памяти включают в свой состав аппаратуру для периодической регенерации ДЗУПВ.
В отличие от динамических, статические ЗУПВ не требуют регенерации и время доступа к ним совпадает с длительностью цикла. Для микросхем, использующих примерно одну и ту же технологию, емкость ДЗУВП по грубым оценкам в 4 - 8 раз превышает емкость СЗУПВ, но последние имеют в 8 - 16 раз меньшую длительность цикла и большую стоимость. По этим причинам в основной памяти практически любого компьютера, проданного после 1975 года, использовались полупроводниковые микросхемы ДЗУПВ (для построения кэш-памяти при этом применялись СЗУПВ). Естественно были и исключения, например, в оперативной памяти суперкомпьютеров компании Cray Research использовались микросхемы СЗУПВ.
Для обеспечения сбалансированности системы с ростом скорости процессоров должна линейно расти и емкость основной памяти. В последние годы емкость микросхем динамической памяти учетверялась каждые три года, увеличиваясь примерно на 60% в год. К сожалению, скорость этих схем за этот же период росла гораздо меньшими темпами (примерно на 7% в год). В то же время производительность процессоров, начиная с 1987 года, практически увеличивалась на 50% в год. На рис. 7.4 представлены основные временные параметры различных поколений ДЗУПВ.

Год появления	Емкость кристалла	Длительность RAS		Длительность CAS	Время цикла	Оптими-зированный режим
		max	min			
1980	64 Кбит	180 нс	150 нс	75 нс	250 нс	150 нс
1983	256 Кбит	150 нс	120 нс	50 нс	220 нс	100 нс
1986	1 Мбит	120 нс	100 нс	25 нс	190 нс	50 нс
1989	4 Мбит	100 нс	80 нс	20 нс	165 нс	40 нс
1992	16 Мбит	80 нс	60 нс	15 нс	120 нс	30 нс
1995	64 Мбит	65 нс	45 нс	10 нс	100 нс	20 нс

Рис. 7.4. Временные параметры ДЗУПВ (в последней строке приведены ожидаемые параметры)

Очевидно, согласование производительности современных процессоров со скоростью основной памяти вычислительных систем остается на сегодняшний день одной из важнейших проблем. Приведенные в предыдущем разделе методы повышения производительности за счет увеличения размеров кэш-памяти и введения многоуровневой организации кэш-памяти могут оказаться не достаточно эффективными с точки зрения стоимости систем. Поэтому важным направлением современных разработок являются методы повышения полосы пропускания или пропускной способности памяти за счет ее организации, включая специальные методы организации ДЗУПВ.
Хотя для организации кэш-памяти в большей степени важно уменьшение задержки памяти, чем увеличение полосы пропускания. Однако при увеличении полосы

пропускания памяти возможно увеличение размера блоков кэш-памяти без заметного увеличения потерь при промахах.

Основными методами увеличения полосы пропускания памяти являются: увеличение разрядности или "ширины" памяти, использование расслоения памяти, использование независимых банков памяти, обеспечение режима бесконфликтного обращения к банкам памяти, использование специальных режимов работы динамических микросхем памяти.

6.3.2. **7.3.2. Увеличение разрядности основной памяти**

Кэш-память первого уровня во многих случаях имеет физическую ширину шин данных соответствующую количеству разрядов в слове, поскольку большинство компьютеров выполняют обращения именно к этой единице информации. В системах без кэш-памяти второго уровня ширина шин данных основной памяти часто соответствует ширине шин данных кэш-памяти. Удвоение или учетверение ширины шин кэш-памяти и основной памяти удваивает или учетверяет соответственно полосу пропускания системы памяти.

Реализация более широких шин вызывает необходимость мультиплексирования данных между кэш-памятью и процессором, поскольку основной единицей обработки данных в процессоре все еще остается слово. Эти мультиплексоры оказываются на критическом пути поступления информации в процессор. Кэш-память второго уровня несколько смягчает эту проблему, т.к. в этом случае мультиплексоры могут располагаться между двумя уровнями кэш-памяти, т.е. вносимая ими задержка не столь критична. Другая проблема, связанная с увеличением разрядности памяти, определяется необходимостью определения минимального объема (инкремента) для поэтапного расширения памяти, которое часто выполняется самими пользователями на месте эксплуатации системы. Удвоение или учетверение ширины памяти приводит к удвоению или учетверению этого минимального инкремента. Наконец, имеются проблемы и с организацией коррекции ошибок в системах с широкой памятью.

Примером организации широкой основной памяти является система Alpha AXP 21064, в которой кэш второго уровня, шина памяти и сама память имеют разрядность в 256 бит.

6.3.3. **7.3.3. Память с расслоением**

Наличие в системе множества микросхем памяти позволяет использовать потенциальный параллелизм, заложенный в такой организации. Для этого микросхемы памяти часто объединяются в банки или модули, содержащие фиксированное число слов, причем только к одному из этих слов банка возможно обращение в каждый момент времени. Как уже отмечалось, в реальных системах имеющаяся скорость доступа к таким банкам памяти редко оказывается достаточной . Следовательно, чтобы получить большую скорость доступа, нужно осуществлять одновременный доступ ко многим банкам памяти. Одна из общих методик, используемых для этого, называется расслоением памяти. При расслоении банки памяти обычно упорядочиваются так, чтобы N последовательных адресов памяти i, i+1, i+2, ..., i+ N-1 приходились на N различных банков. В i-том банке памяти находятся только слова, адреса которых имеют вид $kN + i$ (где $0 \le k \le M-1$, а M число слов в одном банке). Можно достичь в N раз большей скорости доступа к памяти в целом, чем у отдельного ее банка, если обеспечить при каждом доступе обращение к данным в каждом из банков. Имеются разные способы реализации таких расслоенных структур. Большинство из них напоминают конвейеры, обеспечивающие рассылку адресов в различные банки и мультиплексирующие поступающие из банков данные. Таким образом, степень или коэффициент расслоения определяют распределение адресов по банкам памяти. Такие системы оптимизируют обращения по последовательным адресам памяти, что является характерным при подкачке информации в кэш-память при чтении, а также при записи, в случае использования кэш-памятью механизмов обратного копирования. Однако, если требуется доступ к непоследовательно расположенным словам памяти, производительность расслоенной памяти может значительно снижаться.

Обобщением идеи расслоения памяти является возможность реализации нескольких независимых обращений, когда несколько контроллеров памяти позволяют банкам памяти (или группам расслоенных банков памяти) работать независимо.

Если система памяти разработана для поддержки множества независимых запросов (как это имеет место при работе с кэш-памятью, при реализации многопроцессорной и векторной обработки), эффективность системы будет в значительной степени зависеть от частоты поступления независимых запросов к разным банкам. Обращения по последовательным адресам, или в более общем случае, обращения по адресам, отличающимся на нечетное число, хорошо обрабатываются традиционными схемами расслоенной памяти. Проблемы возникают, если разница в адресах последовательных обращений четная. Одно из решений, используемое в больших компьютерах, заключается в том, чтобы статистически уменьшить вероятность подобных обращений путем значительного увеличения количества банков памяти. Например, в суперкомпьютере NEC SX/3 используются 128 банков памяти.

Подобные проблемы могут быть решены как программными, так и аппаратными средствами.

6.3.4. **7.3.4. Использование специфических свойств динамических ЗУПВ**

Как упоминалось раньше, обращение к ДЗУПВ состоит из двух этапов: обращения к строке и обращения к столбцу. При этом внутри микросхемы осуществляется буферизация битов строки, прежде чем происходит обращение к столбцу. Размер строки обычно является корнем квадратным от емкости кристалла памяти: 1024 бита для 1Мбит, 2048 бит для 4 Мбит и т.д. С целью увеличения производительности все современные микросхемы памяти обеспечивают возможность подачи сигналов синхронизации, которые позволяют выполнять последовательные обращения к буферу без дополнительного времени обращения к строке. Имеются три способа подобной оптимизации:

- • блочный режим (nibble mode) - ДЗУВП может обеспечить выдачу четырех последовательных ячеек для каждого сигнала RAS.
- • страничный режим (page mode) - Буфер работает как статическое ЗУПВ; при изменении адреса столбца возможен доступ к произвольным битам в буфере до тех пор, пока не поступит новое обращение к строке или не наступит время регенерации.
- • режим статического столбца (static column) - Очень похож на страничный режим за исключением того, что не обязательно переключать строб адреса столбца каждый раз для изменения адреса столбца.

Начиная с микросхем ДЗУПВ емкостью 1 Мбит, большинство ДЗУПВ допускают любой из этих режимов, причем выбор режима осуществляется на стадии установки кристалла в корпус путем выбора соответствующих соединений. Эти операции изменили определение длительности цикла памяти для ДЗУВП. На рис. 7.4 показано традиционное время цикла и максимальная скорость между обращениями в оптимизированном режиме.

Преимуществом такой оптимизации является то, что она основана на внутренних схемах ДЗУПВ и незначительно увеличивает стоимость системы, позволяя практически учетверить пропускную способность памяти. Например, *nibble mode* был разработан для поддержки режимов, аналогичных расслоению памяти. Кристалл за один раз читает значения четырех бит и подает их наружу в течение четырех оптимизированных циклов. Если время пересылки по шине не превосходит время оптимизированного цикла, единственное усложнение для организации памяти с четырехкратным расслоением заключается в несколько усложненной схеме управления синхросигналами. Страничный режим и режим статического столбца также могут использоваться, обеспечивая даже большую степень расслоения при несколько более сложном управлении. Одной из тенденций в разработке ДЗУПВ является наличие в них буферов с тремя состояниями. Это предполагает, что для реализации традиционного расслоения с большим числом кристаллов памяти в системе должны быть предусмотрены буферные микросхемы для каждого банка памяти.

Новые поколения ДЗУВП разработаны с учетом возможности дальнейшей оптимизации интерфейса между ДЗУПВ и процессором. В качестве примера можно привести изделия компании RAMBUS. Эта компания берет стандартную начинку ДЗУПВ и обеспечивает новый интерфейс, делающий работу отдельной микросхемы более похожей на работу системы памяти, а не на работу отдельного ее компонента. RAMBUS отбросила сигналы RAS/CAS, заменив их шиной, которая допускает выполнение других обращений в интервале между посылкой адреса и приходом данных. Такого рода шины называются шинами с пакетным переключением (packet-switched bus) или шинами с расщепленными транзакциями (split-trancaction bus), которые будут рассмотрены в других главах. Такая шина позволяет работать кристаллу как отдельному банку памяти. Кристалл может вернуть переменное количество данных на один запрос и даже самостоятельно выполняет регенерацию. RAMBUS предлагает байтовый интерфейс и сигнал синхронизации, так что микросхема может тесно синхронизироваться с тактовой частотой процессора. После того, как адресный конвейер наполнен, отдельный кристалл может выдавать по байту каждые 2 нсек.

Большинство систем основной памяти используют методы, подобные страничному режиму ДЗУПВ, для уменьшения различий в производительности процессоров и микросхем памяти.

6.4. 7.4. Виртуальная память и организация защиты памяти

6.4.1. 7.4.1. Концепция виртуальной памяти

Общепринятая в настоящее время концепция виртуальной памяти появилась достаточно давно. Она позволила решить целый ряд актуальных вопросов организации вычислений. Прежде всего к числу таких вопросов относится обеспечение надежного функционирования мультипрограммных систем.

В любой момент времени компьютер выполняет множество процессов или задач, каждая из которых располагает своим адресным пространством. Было бы слишком накладно отдавать всю физическую память какой-то одной задаче тем более, что многие задачи реально используют только небольшую часть своего адресного пространства. Поэтому необходим механизм разделения небольшой физической памяти между различными задачами. Виртуальная память является одним из способов реализации такой возможности. Она делит физическую память на блоки и распределяет их между различными задачами. При этом она предусматривает также некоторую схему защиты, которая ограничивает задачу теми блоками, которые ей принадлежат. Большинство типов виртуальной памяти сокращают также время начального запуска программы на процессоре, поскольку не весь программный код и данные требуются ей в физической памяти, чтобы начать выполнение.

Другой вопрос, тесно связанный с реализацией концепции виртуальной памяти, касается организации вычислений на компьютере задач очень большого объема. Если программа становилась слишком большой для физической памяти, часть ее необходимо было хранить во внешней памяти (на диске) и задача приспособить ее для решения на компьютере ложилась на программиста. Программисты делили программы на части и затем определяли те из них, которые можно было бы выполнять независимо, организуя оверлейные структуры, которые загружались в основную память и выгружались из нее под управлением программы пользователя. Программист должен был следить за тем, чтобы программа не обращалась вне отведенного ей пространства физической памяти. Виртуальная память освободила программистов от этого бремени. Она автоматически управляет двумя уровнями иерархии памяти: основной памятью и внешней (дисковой) памятью.

Кроме того, виртуальная память упрощает также загрузку программ, обеспечивая механизм автоматического перемещения программ, позволяющий выполнять одну и ту же программу в произвольном месте физической памяти.

Системы виртуальной памяти можно разделить на два класса: системы с фиксированным размером блоков, называемых страницами, и системы с переменным размером блоков, называемых сегментами. Ниже рассмотрены оба типа организации виртуальной памяти.

6.4.2. **7.4.2. Страничная организация памяти**

В системах со страничной организацией основная и внешняя память (главным образом дисковое пространство) делятся на блоки или страницы фиксированной длины. Каждому пользователю предоставляется некоторая часть адресного пространства, которая может превышать основную память компьютера, и ограничена только возможностями адресации, заложенными в системе команд. Эта часть адресного пространства называется виртуальной памятью пользователя. Каждое слово в виртуальной памяти пользователя определяется виртуальным адресом, состоящим из двух частей: старшие разряды адреса рассматриваются как номер страницы, а младшие - как номер слова (или байта) внутри страницы.

Управление различными уровнями памяти осуществляется программами ядра операционной системы, которые следят за распределением страниц и оптимизируют обмены между этими уровнями. При страничной организации памяти смежные виртуальные страницы не обязательно должны размещаться на смежных страницах основной физической памяти. Для указания соответствия между виртуальными страницами и страницами основной памяти операционная система должна сформировать таблицу страниц для каждой программы и разместить ее в основной памяти машины. При этом каждой странице программы, независимо от того находится ли она в основной памяти или нет, ставится в соответствие некоторый элемент таблицы страниц. Каждый элемент таблицы страниц содержит номер физической страницы основной памяти и специальный индикатор. Единичное состояние этого индикатора свидетельствует о наличии этой страницы в основной памяти. Нулевое состояние индикатора означает отсутствие страницы в оперативной памяти.

Для увеличения эффективности такого типа схем в процессорах используется специальная полностью ассоциативная кэш-память, которая также называется буфером преобразования адресов (TLB tranclation-lookaside buffer). Хотя наличие TLB не меняет принципа построения схемы страничной организации, с точки зрения защиты памяти, необходимо предусмотреть возможность очистки его при переключении с одной программы на другую.

Поиск в таблицах страниц, расположенных в основной памяти, и загрузка TLB может осуществляться либо программным способом, либо специальными аппаратными средствами. В последнем случае для того, чтобы предотвратить возможность обращения пользовательской программы к таблицам страниц, с которыми она не связана, предусмотрены специальные меры. С этой целью в процессоре предусматривается дополнительный регистр защиты, содержащий описатель (дескриптор) таблицы страниц или базово-граничную пару. База определяет адрес начала таблицы страниц в основной памяти, а граница - длину таблицы страниц

соответствующей программы. Загрузка этого регистра защиты разрешена только в привилегированном режиме. Для каждой программы операционная система хранит дескриптор таблицы страниц и устанавливает его в регистр защиты процессора перед запуском соответствующей программы.
Отметим некоторые особенности, присущие простым схемам со страничной организацией памяти. Наиболее важной из них является то, что все программы, которые должны непосредственно связываться друг с другом без вмешательства операционной системы, должны использовать общее пространство виртуальных адресов. Это относится и к самой операционной системе, которая, вообще говоря, должна работать в режиме динамического распределения памяти. Поэтому в некоторых системах пространство виртуальных адресов пользователя укорачивается на размер общих процедур, к которым программы пользователей желают иметь доступ. Общим процедурам должен быть отведен определенный объем пространства виртуальных адресов всех пользователей, чтобы они имели постоянное место в таблицах страниц всех пользователей. В этом случае для обеспечения целостности, секретности и взаимной изоляции выполняющихся программ должны быть предусмотрены различные режимы доступа к страницам, которые реализуются с помощью специальных индикаторов доступа в элементах таблиц страниц.
Следствием такого использования является значительный рост таблиц страниц каждого пользователя. Одно из решений проблемы сокращения длины таблиц основано на введении многоуровневой организации таблиц. Частным случаем многоуровневой организации таблиц является сегментация при страничной организации памяти. Необходимость увеличения адресного пространства пользователя объясняется желанием избежать необходимости перемещения частей программ и данных в пределах адресного пространства, которые обычно приводят к проблемам переименования и серьезным затруднениям в разделении общей информации между многими задачами.

6.4.3. **7.4.3. Сегментация памяти**

Другой подход к организации памяти опирается на тот факт, что программы обычно разделяются на отдельные области-сегменты. Каждый сегмент представляет собой отдельную логическую единицу информации, содержащую совокупность данных или программ и расположенную в адресном пространстве пользователя. Сегменты создаются пользователями, которые могут обращаться к ним по символическому имени. В каждом сегменте устанавливается своя собственная нумерация слов, начиная с нуля.
Обычно в подобных системах обмен информацией между пользователями строится на базе сегментов. Поэтому сегменты являются отдельными логическими единицами информации, которые необходимо защищать, и именно на этом уровне вводятся различные режимы доступа к сегментам. Можно выделить два основных типа сегментов: программные сегменты и сегменты данных (сегменты стека являются частным случаем сегментов данных). Поскольку общие программы должны обладать свойством повторной входимости, то из программных сегментов допускается только выборка команд и чтение констант. Запись в программные сегменты может рассматриваться как незаконная и запрещаться системой. Выборка команд из сегментов данных также может считаться незаконной и любой сегмент данных может быть защищен от обращений по записи или по чтению.
Для реализации сегментации было предложено несколько схем, которые отличаются деталями реализации, но основаны на одних и тех же принципах.
В системах с сегментацией памяти каждое слово в адресном пространстве пользователя определяется виртуальным адресом, состоящим из двух частей: старшие разряды адреса рассматриваются как номер сегмента, а младшие - как номер слова внутри сегмента. Наряду с сегментацией может также использоваться страничная

организация памяти. В этом случае виртуальный адрес слова состоит из трех частей: старшие разряды адреса определяют номер сегмента, средние - номер страницы внутри сегмента, а младшие - номер слова внутри страницы.

Как и в случае страничной организации, необходимо обеспечить преобразование виртуального адреса в реальный физический адрес основной памяти. С этой целью для каждого пользователя операционная система должна сформировать таблицу сегментов. Каждый элемент таблицы сегментов содержит описатель (дескриптор) сегмента (поля базы, границы и индикаторов режима доступа). При отсутствии страничной организации поле базы определяет адрес начала сегмента в основной памяти, а граница - длину сегмента. При наличии страничной организации поле базы определяет адрес начала таблицы страниц данного сегмента, а граница - число страниц в сегменте. Поле индикаторов режима доступа представляет собой некоторую комбинацию признаков блокировки чтения, записи и выполнения.

Таблицы сегментов различных пользователей операционная система хранит в основной памяти. Для определения расположения таблицы сегментов выполняющейся программы используется специальный регистр защиты, который загружается операционной системой перед началом ее выполнения. Этот регистр содержит дескриптор таблицы сегментов (базу и границу), причем база содержит адрес начала таблицы сегментов выполняющейся программы, а граница - длину этой таблицы сегментов. Разряды номера сегмента виртуального адреса используются в качестве индекса для поиска в таблице сегментов. Таким образом, наличие базово-граничных пар в дескрипторе таблицы сегментов и элементах таблицы сегментов предотвращает возможность обращения программы пользователя к таблицам сегментов и страниц, с которыми она не связана. Наличие в элементах таблицы сегментов индикаторов режима доступа позволяет осуществить необходимый режим доступа к сегменту со стороны данной программы. Для повышения эффективности схемы используется ассоциативная кэш-память.

Отметим, что в описанной схеме сегментации таблица сегментов с индикаторами доступа предоставляет всем программам, являющимся частями некоторой задачи, одинаковые возможности доступа, т. е. она определяет единственную область (домен) защиты. Однако для создания защищенных подсистем в рамках одной задачи для того, чтобы изменять возможности доступа, когда точка выполнения переходит через различные программы, управляющие ее решением, необходимо связать с каждой задачей множество доменов защиты. Реализация защищенных подсистем требует разработки некоторых специальных аппаратных средств. Рассмотрение таких систем, которые включают в себя кольцевые схемы защиты, а также различного рода мандатные схемы защиты, выходит за рамки данного обзора.

8. 8. Современные микропроцессоры

8.1. 8.1. Процессоры с архитектурой 80x86 и Pentium

Обычно, когда новая архитектура создается одним архитектором или группой архитекторов, ее отдельные части очень хорошо подогнаны друг к другу и вся архитектура может быть описана достаточно связано. Этого нельзя сказать об архитектуре 80x86, поскольку это продукт нескольких независимых групп разработчиков, которые развивали эту архитектуру более 15 лет, добавляя к первоначальному набору команд новые возможности.

В 1978 году была анонсирована архитектура Intel 8086 как совместимое вверх расширение в то время успешного 8-бит микропроцессора 8080. 8086 представляет собой 16-битовую архитектуру со всеми внутренними регистрами, имеющими 16-битовую разрядность. Микропроцессор 8080 был просто построен на базе накапливающего сумматора (аккумулятора), но архитектура 8086 была расширена дополнительными регистрами. Поскольку почти каждый регистр в этой архитектуре имеет определенное назначение, 8086 по классификации частично можно отнести к машинам с накапливающим сумматором, а частично - к машинам с регистрами общего назначения, и его можно назвать расширенной машиной с накапливающим сумматором. Микропроцессор 8086 (точнее его версия 8088 с 8-битовой внешней шиной) стал основой завоевавшей в последствии весь мир серии компьютеров IBM PC, работающих под управлением операционной системы MS-DOS.

В 1980 году был анонсирован сопроцессор плавающей точки 8087. Эта архитектура расширила 8086 почти на 60 команд плавающей точки. Ее архитекторы отказались от расширенных накапливающих сумматоров для того, чтобы создать некий гибрид стеков и регистров, по сути расширенную стековую архитектуру. Полный набор стековых команд дополнен ограниченным набором команд типа регистр-память.

Анонсированный в 1982 году микропроцессор 80286, еще дальше расширил архитектуру 8086. Была создана сложная модель распределения и защиты памяти, расширено адресное пространство до 24 разрядов, а также добавлено небольшое число дополнительных команд. Поскольку очень важно было обеспечить выполнение без изменений программ, разработанных для 8086, в 80286 был предусмотрен режим реальных адресов, позволяющий машине выглядеть почти как 8086. В 1984 году компания IBM объявила об использовании этого процессора в своей новой серии персональных компьютеров IBM PC/AT.

В 1987 году появился микропроцессор 80386, который расширил архитектуру 80286 до 32 бит. В дополнение к 32-битовой архитектуре с 32-битовыми регистрами и 32-битовым адресным пространством, в микропроцессоре 80386 появились новые режимы адресации и дополнительные операции. Все эти расширения превратили 80386 в машину, по идеологии близкую к машинам с регистрами общего назначения. В дополнение к механизмам сегментации памяти, в микропроцессор 80386 была добавлена также поддержка страничной организации памяти. Также как и 80286, микропроцессор 80386 имеет режим выполнения программ, написанных для 8086. Хотя в то время базовой операционной системой для этих микропроцессоров оставалась MS-DOS, 32-разрядная архитектура и страничная организация памяти послужили основой для переноса на эту платформу операционной системы UNIX. Следует отметить, что для процессора 80286 была создана операционная система XENIX (сильно урезанный вариант системы UNIX).

Эта история иллюстрирует эффект, вызванный необходимостью обеспечения совместимости с 80x86, поскольку существовавшая база программного обеспечения на каждом шаге была слишком важной. К счастью, последующие процессоры (80486 в 1989 и Pentium в 1993 году) были нацелены на увеличение производительности и добавили к видимому пользователем набору команд только три новые команды, облегчающие организацию многопроцессорной работы.
Что бы ни говорилось о неудобствах архитектуры 80x86, следует иметь в виду, что она преобладает в мире персональных компьютеров. Почти 80% установленных малых систем базируются именно на этой архитектуре. Споры относительно преимуществ CISC и RISC архитектур постепенно стихают, поскольку современные микропроцессоры стараются вобрать в себя наилучшие свойства обоих подходов.
Современное семейство процессоров i486 (i486SX, i486DX, i486DX2 и i486DX4), в котором сохранились система команд и методы адресации процессора i386, уже имеет некоторые свойства RISC-микропроцессоров. Например, наиболее употребительные команды выполняются за один такт. Компания Intel для оценки производительности своих процессоров ввела в употребление специальную характеристику, которая называется рейтингом iCOMP. Компания надеялась, что эта характеристика станет стандартной тестовой оценкой и будет применяться другими производителями микропроцессоров, однако последние с понятной осторожностью относятся к системе измерений производительности, введенной компанией Intel. Ниже в таблице приведены сравнительные характеристики некоторых процессоров компании Intel на базе рейтинга iCOMP.

Процессор	Тактовая частота (МГц)	Рейтинг iCOMP
386SX	25	39
386SL	25	41
386DX	25	49
386DX	33	68
i486SX	20	78
i486SX	25	100
i486SX	33	136
i486DX	33	166
i486DX2	50	231
i486DX	50	249
i486DX2	66	297
i486DX4	75	319
i486DX4	100	435
Pentium	60	510
Pentium	66	567
Pentium	90	735
Pentium	100	815
Pentium	120	1000
Pentium	133	1200

Процессоры i486SX и i486DX - это 32-битовые процессоры с внутренней кэш-па-мятью емкостью 8 Кбайт и 32-битовой шиной данных. Основное отличие между ними заключается в том, что в процессоре i486SX отсутствует интегрированный сопроцессор плавающей точки. Поэтому он имеет меньшую цену и применяется в системах, для которых не очень важна производительность при обработке вещественных чисел. Для этих систем обычно возможно расширение с помощью внешнего сопроцессора i487SX.
Процессоры Intel OverDrive и i486DX2 практически идентичны. Однако кристалл OverDrive имеет корпус, который может устанавливаться в гнездо расширения сопроцессора i487SX, применяемое в ПК на базе i486SX. В процессорах OverDrive и i486DX2 применяется технология удвоения внутренней тактовой частоты, что позволяет

увеличить производительность процессора почти на 70%. Процессор i486DX4/100 использует технологию утроения тактовой частоты. Он работает с внутренней тактовой частотой 99 МГц, в то время как внешняя тактовая частота (частота, на которой работает внешняя шина) составляет 33 МГц. Этот процессор практически обеспечивает равные возможности с машинами класса 60 МГц Pentium, являясь их полноценной и доступной по цене альтернативой.
Появившийся в 1993 году процессор Pentium ознаменовал собой новый этап в развитии архитектуры x86, связанный с адаптацией многих свойств процессоров с архитектурой RISC. Он изготовлен по 0.8 микронной БиКМОП технологии и содержит 3.1 миллиона транзисторов. Первоначальная реализация была рассчитана на работу с тактовой частотой 60 и 66 МГц. В настоящее время имеются также процессоры Pentium, работающие с тактовой частотой 75, 90, 100, 120, 133, 150 и 200 МГц. Процессор Pentium по сравнению со своими предшественниками обладает целым рядом улучшенных характеристик. Главными его особенностями являются:

- • двухпотоковая суперскалярная организация, допускающая параллельное выполнение пары простых команд;
- • наличие двух независимых двухканальных множественно-ассоциативных кэшей для команд и для данных, обеспечивающих выборку данных для двух операций в каждом такте;
- • динамическое прогнозирование переходов;
- • конвейерная организация устройства плавающей точки с 8 ступенями;
- • двоичная совместимость с существующими процессорами семейства 80x86.

Блок-схема процессора Pentium представлена на рисунке 8.1. Прежде всего, новая микроархитектура этого процессора базируется на идее суперскалярной обработки (правда, с некоторыми ограничениями). Основные команды распределяются по двум независимым исполнительным устройствам (конвейерам U и V). Конвейер U может выполнять любые команды семейства x86, включая целочисленные команды и команды с плавающей точкой. Конвейер V предназначен для выполнения простых целочисленных команд и некоторых команд с плавающей точкой. Команды могут направляться в каждое из этих устройств одновременно, причем при выдаче устройством управления в одном такте пары команд более сложная команда поступает в конвейер U, а менее сложная - в конвейер V. Такая попарная выдача команд возможна, правда, только для ограниченного подмножества целочисленных команд. Команды арифметики с плавающей точкой не могут запускаться в паре с целочисленными командами. Одновременная выдача двух команд возможна только при отсутствии зависимостей по регистрам. При остановке команды по любой причине в одном конвейере, как правило, останавливается и второй конвейер.
Остальные устройства процессора предназначены для снабжения конвейеров необходимыми командами и данными. В отличие от процессоров i486 в процессоре Pentium используется раздельная кэш-память команд и данных емкостью по 8 Кбайт, что обеспечивает независимость обращений. За один такт из каждой кэш-памяти могут считываться два слова. При этом кэш-память данных построена на принципах двухкратного расслоения, что обеспечивает одновременное считывание двух слов, принадлежащих одной строке кэш-памяти. Кэш-память команд хранит сразу три копии тегов, что позволяет в одном такте считывать два командных слова, принадлежащих либо одной строке, либо смежным строкам для обеспечения попарной выдачи команд, при этом третья копия тегов используется для организации протокола наблюдения за когерентностью состояния кэш-памяти. Для повышения эффективности перезагрузки кэш-памяти в процессоре применяется 64-битовая внешняя шина данных.

В процессоре предусмотрен механизм динамического прогнозирования направления переходов. С этой целью на кристалле размещена небольшая кэш-память, которая называется буфером целевых адресов переходов (BTB), и две независимые пары буферов предварительной выборки команд (по два 32-битовых буфера на каждый конвейер). Буфер целевых адресов переходов хранит адреса команд, которые находятся в буферах предварительной выборки. Работа буферов предварительной выборки организована таким образом, что в каждый момент времени осуществляется выборка команд только в один из буферов соответствующей пары. При обнаружении в потоке команд операции перехода вычисленный адрес перехода сравнивается с адресами, хранящимися в буфере BTB. В случае совпадения предсказывается, что переход будет выполнен, и разрешается работа другого буфера предварительной выборки, который начинает выдавать команды для выполнения в соответствующий конвейер. При несовпадении считается, что переход выполняться не будет, и буфер предварительной выборки не переключается, продолжая обычный порядок выдачи команд. Это позволяет избежать простоев конвейеров при правильном прогнозе направления перехода. Окончательное решение о направлении перехода естественно принимается на основании анализа кода условия. При неправильно сделанном прогнозе содержимое конвейеров аннулируется, и выдача команд начинается с необходимого адреса. Неправильный прогноз приводит к приостановке работы конвейеров на 3-4 такта.

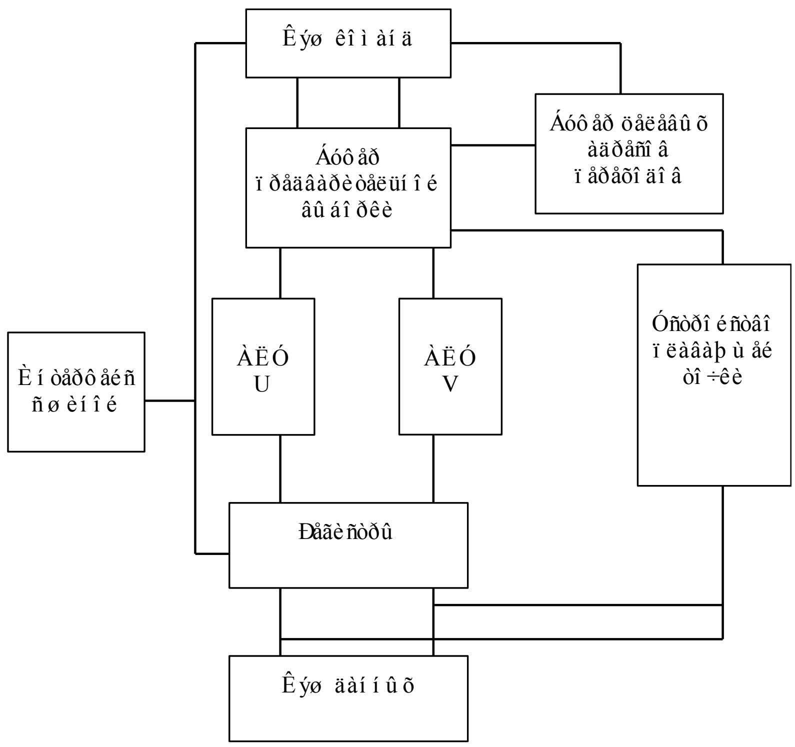


Рис. 8.1. Упрощенная блок схема процессора Pentium

Следует отметить, что возросшая производительность процессора Pentium требует и соответствующей организации системы на его основе. Компания Intel разработала и поставляет все необходимые для этого наборы микросхем. Прежде всего, для согласования скорости с динамической основной памятью необходима кэш-память второго уровня. Контроллер кэш-памяти 82496 и микросхемы статической памяти 82491 обеспечивают построение такой кэш-памяти объемом 256 Кбайт и работу процессора без тактов ожидания. Для эффективной организации систем Intel разработала стандарт на высокопроизводительную локальную шину PCI. Выпускаются наборы микросхем для построения мощных компьютеров на ее основе.

В 1995 году компания Intel разработала и выпустила новый процессор, продолжающий архитектурную линию x86. Этот процессор получил название P6 или PentiumPro. Он работает с тактовыми частотами 150: 166: 180 и 200 МГц. PentiumPro обеспечивает полную совместимость с процессорами предыдущих поколений. Он предназначен главным образом для поддержки высокопроизводительных 32-битовых вычислений в области САПР, трехмерной графики и мультимедиа: а также широкого круга коммерческих приложений баз данных. По результатам испытаний на тестах SPEC (8.58 SPECint95 и 6.48 SPECfp95) процессор PentiumPro по производительности целочисленных операций в 1996 году вышел на третье место в мировой классификации, уступая только 180 МГц HP PA-8000 и 400 МГц DEC Alpha (см. рис. 8.2). Для достижения такой производительности необходимо использование технических решений, широко применяющихся при построении RISC-процессоров:

- • выполнение команд не в предписанной программой последовательности, что устраняет во многих случаях приостановку конвейеров из-за ожидания операндов операций;
- • использование методики переименования регистров, позволяющей увеличивать эффективный размер регистрового файла (малое количество регистров - одно из самых узких мест архитектуры x86);
- • расширение суперскалярных возможностей по отношению к процессору Pentium, в котором обеспечивается одновременная выдача только двух команд с достаточно жесткими ограничениями на их комбинации.

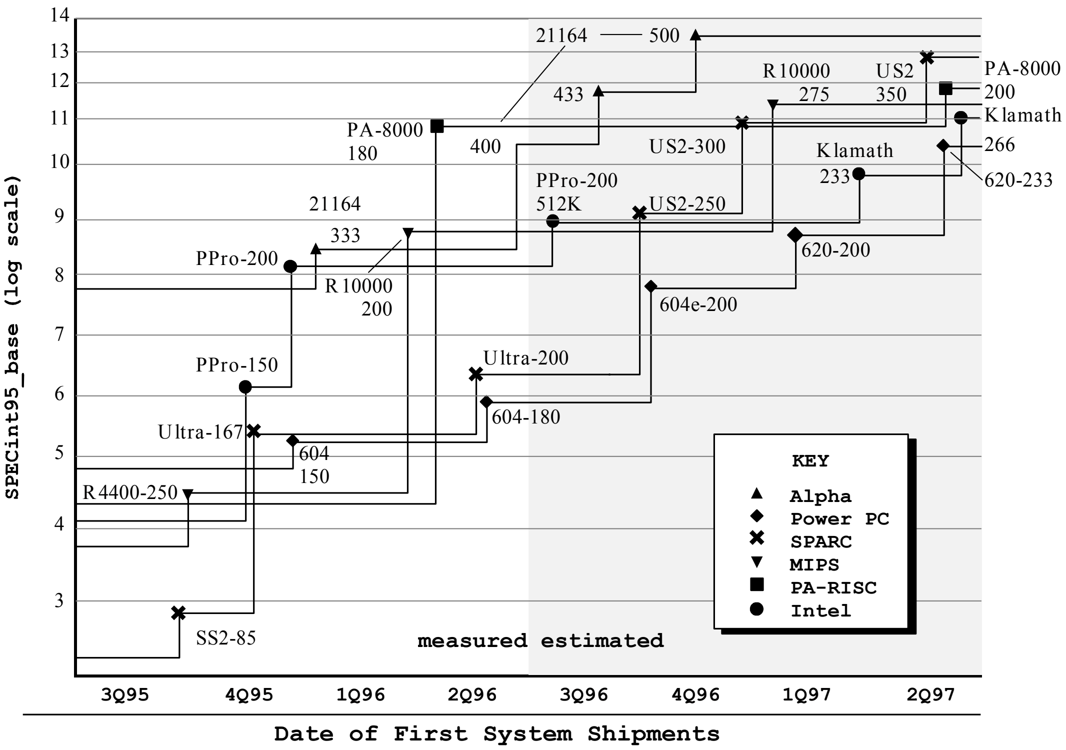


Рис. 8.2.

Кроме того, в борьбу за новое поколение процессоров x86 включились компании, ранее занимавшиеся изготовлением Intel-совместимых процессоров. Это компании Advanced Micro Devices (AMD), Cyrix Corp и NexGen. С точки зрения микроархитектуры наиболее близок к Pentium процессор M1 компании Cyrix. Также как и Pentium он имеет два конвейера и может выполнять до двух команд в одном такте. Однако в процессоре M1 число случаев, когда операции могут выполняться попарно, значительно увеличено. Кроме того, в нем применяется методика обходов и ускорения пересылки данных, позволяющая устранить приостановку конвейеров во многих ситуациях, с которыми не справляется Pentium. Процессор содержит 32 физических регистра (вместо 8 логических, предусмотренных архитектурой x86) и применяет методику переименования регистров для устранения зависимостей по данным. Как и Pentium, процессор M1 для прогнозирования направления перехода использует буфер целевых адресов перехода емкостью 256 элементов, но кроме того поддерживает специальный стек возвратов, отслеживающий вызовы процедур и последующие возвраты.

Процессоры K5 компании AMD и Nx586 компании NexGen используют в корне другой подход. Основа их процессоров - очень быстрое RISC-ядро, выполняющее высокорегулярные операции в суперскалярном режиме. Внутренние форматы команд (ROP у компании AMD и RISC86 у компании NexGen) соответствуют традиционным системам команд RISC-процессоров. Все команды имеют одинаковую длину и кодируются в регулярном формате. Обращения к памяти выполняются специальными командами загрузки и записи. Как известно, архитектура x86 имеет очень сложную для декодирования систему команд. В процессорах K5 и Nx586 осуществляется аппаратная трансляция команд x86 в команды внутреннего формата, что дает лучшие условия для распараллеливания вычислений. В процессоре K5 имеются 40, а в процессоре Nx586 22 физических регистра, которые реализуют методику переименования. В процессоре K5 информация, необходимая для прогнозирования направления перехода, записывается прямо в кэш команд и хранится вместе с каждой строкой кэш-памяти. В процессоре Nx586 для этих целей используется кэш-память адресов переходов на 96 элементов.

Таким образом, компания Intel больше не обладает монополией на методы конструирования высокопроизводительных процессоров x86, и можно ожидать появления новых процессоров, не только не уступающих, но и возможно превосходящих по производительности процессоры компании, стоявшей у истоков этой архитектуры. Следует отметить, что сама компания Intel заключила стратегическое соглашение с компанией Hewlett-Packard на разработку следующего поколения микропроцессоров, в которых архитектура x86 будет сочетаться с архитектурой очень длинного командного слова (VLIW -архитектурой). Появление этих микропроцессоров не ожидается до конца 1998 года.

8.2 8.2 Особенности процессоров с архитектурой SPARC компании Sun Microsystems

Масштабируемая процессорная архитектура SPARC (Scalable Processor Architecture) компании Sun Microsystems является наиболее широко распространенной RISC-архитектурой, отражающей доминирующее положение компании на рынке UNIX рабочих станций и серверов. Процессоры с архитектурой SPARC лицензированы и изготавливаются по спецификациям Sun несколькими производителями, среди которых следует отметить компании Texas Instruments, Fujitsu, LSI Logic, Bipolar International Technology, Philips, Cypress Semiconductor и Ross Technologies. Эти компании осуществляют поставки процессоров SPARC не только самой Sun Microsystems, но и другим известным производителям вычислительных систем, например, Solbourne, Toshiba, Matsushita, Tatung и Cray Research.

Первоначально архитектура SPARC была разработана с целью упрощения реализации 32-битового процессора. В последствии, по мере улучшения технологии изготовления интегральных схем, она постепенно развивалось и в настоящее время имеется 64-битовая версия этой архитектуры (SPARC-V9), которая положена в основу новых микропроцессоров, получивших название UltraSPARC.
Первый процессор SPARC был изготовлен компанией Fujitsu на базе вентильной матрицы, работающей на частоте 16.67 МГц. На основе этого процессора была разработана первая рабочая станция Sun-4 с производительностью 10 MIPS, объявленная осенью 1987 года (до этого времени компания Sun использовала в своих изделиях микропроцессоры Motorola 680X0). В марте 1988 года Fujitsu увеличила тактовую частоту до 25 МГц создав процессор с производительностью 15 MIPS.

Позднее компания Sun умело использовала конкуренцию среди компаний-поставщиков интегральных схем, выбирая наиболее удачные разработки для реализации своих изделий SPARCstation 1, 1+, IPC, ELC, IPX, 2 и серверов серий 4xx и 6xx. Тактовая частота процессоров SPARC была повышена до 40 МГц, а производительность - до 28 MIPS.
Дальнейшее увеличение производительности процессоров с архитектурой SPARC было достигнуто за счет реализации в кристаллах принципов суперскалярной обработки компаниями Texas Instruments и Cypress. Процессор SuperSPARC компании Texas Instruments стал основой серии рабочих станций и серверов SPARCstation/SPARCserver 10 и 20. В зависимости от смеси команд он обеспечивает выдачу до трех команд за один машинный такт. Процессор SuperSPARC имеет сбалансированную производительность на операциях с фиксированной и плавающей точкой. Он имеет внутренний кэш емкостью 36 Кб (20 Кб - кэш команд и 16 Кб - кэш данных), раздельные конвейеры целочисленной и вещественной арифметики и при тактовой частоте 75 МГц обеспечивает производительность около 205 MIPS.
Компания Texas Instruments разработала также 50 МГц процессор MicroSPARC с встроенным кэшем емкостью 6 Кб, который ранее широко использовался в дешевых моделях рабочих станций SPARCclassic и LX. Затем Sun совместно с Fujitsu создали новую версию кристалла MicroSPARC II с встроенным кэшем емкостью 24 Кб. На его основе построены рабочие станции и серверы SPARCstation/SPARCserver 4 и 5, работающие на частоте 70, 85 и 110 МГц.
Хотя архитектура SPARC в течение длительного времени оставалась доминирующей на рынке процессоров RISC, особенно в секторе рабочих станций, повышение тактовой частоты процессоров в 1992-1994 годах происходило более медленными темпами по сравнению с повышением тактовой частоты конкурирующих архитектур процессоров. Чтобы ликвидировать это отставание, а также в ответ на появление на рынке 64-битовых процессоров компания Sun разработала и проводит в жизнь пятилетнюю программу модернизации. В соответствии с этой программой Sun планировала довести тактовую частоту процессоров MicroSPARC до 100 МГц в 1994 году (процессор MicroSPARC II с тактовой частотой 110 МГц используется в рабочих станциях и серверах SPARCstation 4 и 5). В конце 1994 и в течение 1995 года на рынке появились микропроцессоры hyperSPARC и однопроцессорные и многопроцессорные рабочие станции SPARCstation 20 с тактовой частотой процессора 100, 125 и 150 МГц. К середине 1995 года тактовая частота процессоров SuperSPARC была доведена до 85 МГц (60, 75 и 85 МГц версии этого процессора в настоящее время применяются в рабочих станциях и серверах SPARCstation 20, SPARCserver 1000 и SPARCcenter 2000 компании Sun и 64-процессорном сервере компании Cray Research). Наконец, в ноябре 1995 года, появились 64-битовые процессоры UltraSPARC-I с тактовой частотой 143, 167 и 200 МГц, и были объявлены процессоры UltraSPARC-II с тактовой частотой от 250 до 300 МГц, серийное производство которых должно начаться в середине 1996 года. В дальнейшем планируется выпуск процессоров UltraSPARC-III с частотой до 500 МГц.

Таким образом, компания Sun Microsystems в настоящее время обладает широчайшим спектром процессоров, способных удовлетворить нужды практически любого пользователя, как с точки зрения производительности выпускаемых ею рабочих станций и серверов, так и в отношении их стоимости, и судя по всему не собирается уступать своих позиций на быстро меняющемся компьютерном рынке.

8.2.1. **8.2.1. SuperSPARC**

Имеется несколько версий этого процессора, позволяющего в зависимости от смеси команд обрабатывать до трех команд за один машинный такт, отличающихся тактовой частотой (50, 60, 75 и 85 МГц). Процессор SuperSPARC (рисунок 8.3) имеет сбалансированную производительность на операциях с фиксированной и плавающей точкой. Он имеет внутренний кэш емкостью 36 Кб (20 Кб - кэш команд и 16 Кб - кэш данных), раздельные конвейеры целочисленной и вещественной арифметики и при тактовой частоте 75 МГц обеспечивает производительность около 205 MIPS. Процессор SuperSPARC применяется также в серверах SPARCserver 1000 и SPARCcenter 2000 компании Sun.

Конструктивно кристалл монтируется на взаимозаменяемых процессорных модулях трех типов, отличающихся наличием и объемом кэш-памяти второго уровня и тактовой частотой. Модуль M-bus SuperSPARC, используемый в модели 50 содержит 50-МГц SuperSPARC процессор с внутренним кэшем емкостью 36 Кб (20 Кб кэш команд и 16 Кб кэш данных). Модули M-bus SuperSPARC в моделях 51, 61 и 71 содержат по одному SuperSPARC процессору, работающему на частоте 50, 60 и 75 МГц соответственно, одному кристаллу кэш-контроллера (так называемому SuperCache), а также внешний кэш емкостью 1 Мб. Модули M-bus в моделях 502, 612, 712 и 514 содержат два SuperSPARC процессора и два кэш-контроллера каждый, а последние три модели и по одному 1 Мб внешнему кэшу на каждый процессор. Использование кэш-памяти позволяет модулям CPU работать с тактовой частотой, отличной от тактовой частоты материнской платы; пользователи всех моделей поэтому могут улучшить производительность своих систем заменой существующих модулей CPU вместо того, чтобы производить upgrade всей материнской платы.

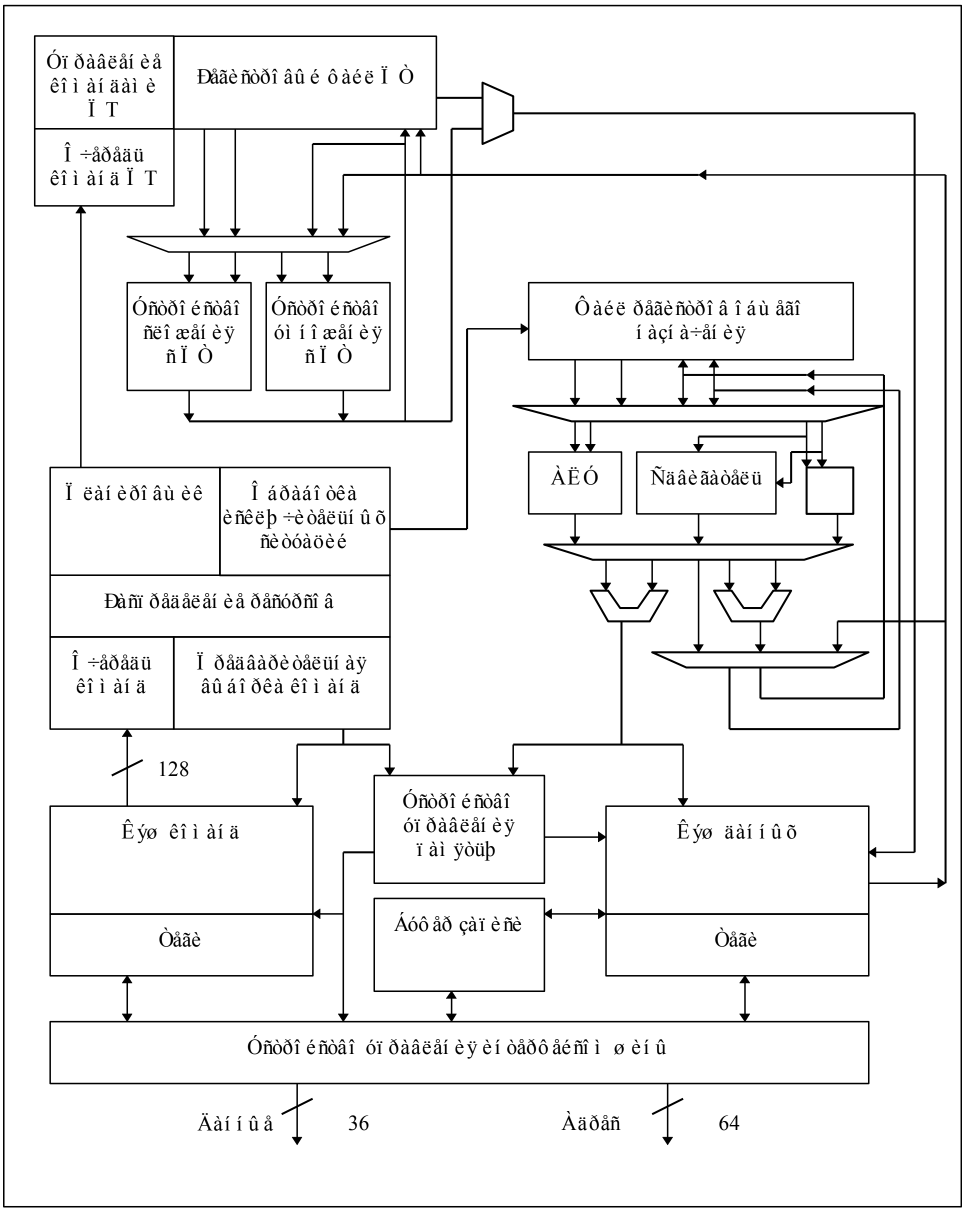


Рис. 8.3. Блок-схема процессора Super SPARC

8.2.2. **8.2.2. hyperSPARC**

Одной из главных задач, стоявших перед разработчиками микропроцессора hyperSPARC, было повышение производительности, особенно при выполнении операций с плавающей точкой. Поэтому особое внимание разработчиков было уделено созданию простых и сбалансированных шестиступенчатых конвейеров целочисленной арифметики и плавающей точки. Логические схемы этих конвейеров тщательно разрабатывались, количество логических уровней вентилей между ступенями выравнивалось, чтобы упростить вопросы дальнейшего повышения тактовой частоты.

Производительность процессоров hyperSPARC может меняться независимо от скорости работы внешней шины (MBus). Набор кристаллов hyperSPARC обеспечивает как синхронные, так и асинхронные операции с помощью специальной логики кристалла RT625. Отделение внутренней шины процессора от внешней шины позволяет увеличивать тактовую частоту процессора независимо от частоты работы подсистем памяти и ввода/вывода. Это обеспечивает более длительный жизненный цикл, поскольку переход на более производительные модули hyperSPARC не требует переделки всей системы.

Процессорный набор hyperSPARC с тактовой частотой 100 МГц построен на основе технологического процесса КМОП с тремя уровнями металлизации и проектными нормами 0.5 микрон. Внутренняя логика работает с напряжением питания 3.3В.

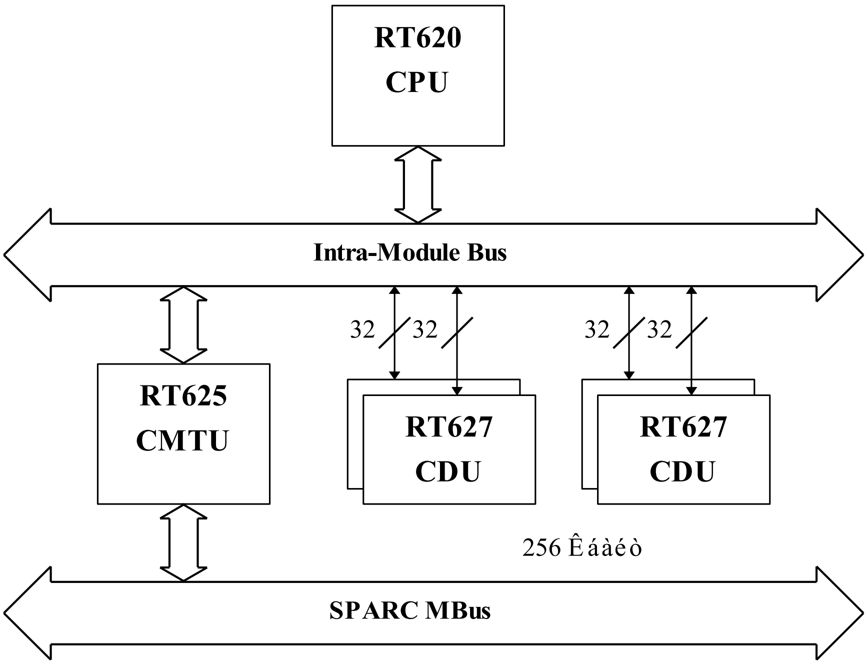


Рис. 8.4. Набор кристаллов процессора hyperSPARC

Процессор hyperSPARC реализован в виде многокристальной микросборки (рисунок 8.4), в состав которой входит суперскалярная конвейерная часть и тесно связанная с ней кэш-память второго уровня. В набор кристаллов входят RT620 (CPU) - центральный процессор, RT625 (CMTU) - контроллер кэш-памяти, устройство управления памятью и устройство тегов и четыре RT627 (CDU) кэш-память данных для реализации кэш-памяти второго уровня емкостью 256 Кбайт. RT625 обеспечивает также интерфейс с MBus.

Центральный процессор RT620 (рисунок 8.5) состоит из целочисленного устройства, устройства с плавающей точкой, устройства загрузки/записи, устройства переходов и двухканальной множественно-ассоциативной памяти команд емкостью 8 Кбайт. Целочисленное устройство включает АЛУ и отдельный тракт данных для операций загрузки/записи, которые представляют собой два из четырех исполнительных устройств процессора. Устройство переходов обрабатывает команды передачи управления, а устройство плавающей точки, реально состоит из двух независимых конвейеров - сложения и умножения чисел с плавающей точкой. Для увеличения пропускной способности процессора команды плавающей точки, проходя через целочисленный конвейер, поступают в очередь, где они ожидают запуска в одном из конвейеров плавающей точки. В каждом такте выбираются две команды. В общем случае, до тех пор, пока эти две команды требуют для своего выполнения различных исполнительных устройств при отсутствии зависимостей по данным, они могут запускаться одновременно. RT620 содержит два регистровых файла: 136

целочисленных регистров, сконфигурированных в виде восьми регистровых окон, и 32 отдельных регистра плавающей точки, расположенных в устройстве плавающей точки.

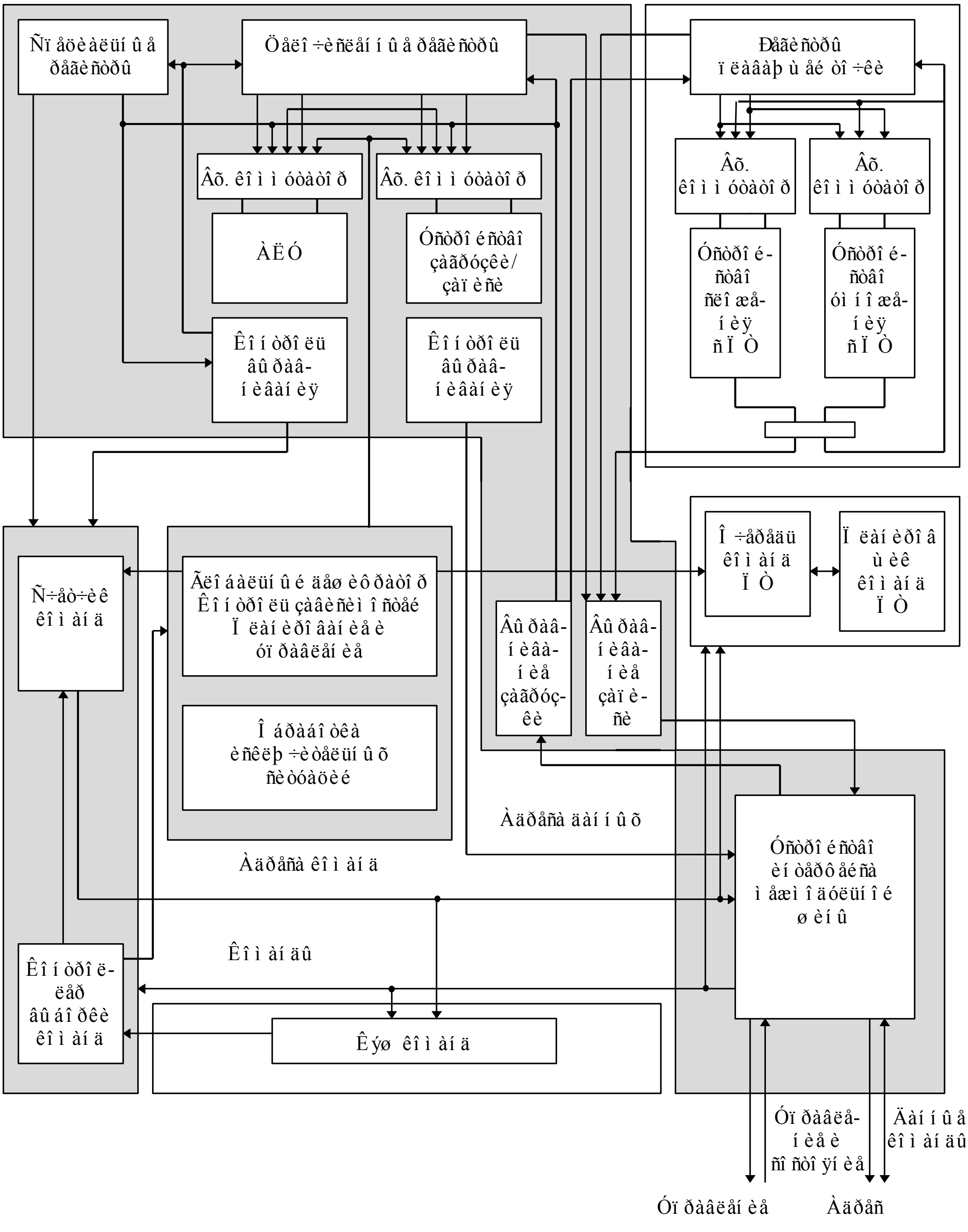


Рис. 8.5. Процессор RT 620

Кэш-память второго уровня в процессоре hyperSPARC строится на базе RT625 CMTU, который представляет собой комбинированный кристалл, включающий контроллер кэш-памяти и устройство управления памятью, которое поддерживает разделяемую внешнюю память и симметричную многопроцессорную обработку. Контроллер кэш-памяти поддерживает кэш емкостью 256 Кбайт, состоящий из четырех RT627 CDU. Кэш-память имеет прямое отображение и 4К тегов. Теги в кэш-памяти содержат физические адреса, поэтому логические схемы для соблюдения когерентности кэш-памяти в многопроцессорной системе, имеющиеся в RT625, могут быстро определить попадания или промахи при просмотре со стороны внешней шины без приостановки обращений к кэш-памяти со стороны центрального процессора. Поддерживается как режим сквозной записи, так и режим обратного копирования.

Устройство управления памятью содержит в своем составе полностью ассоциативную кэш-память преобразования виртуальных адресов в физические (TLB), состоящую из 64 строк, которая поддерживает 4096 контекстов. RT625 содержит буфер чтения емкостью 32 байта, используемый для загрузки, и буфер записи емкостью 64 байта, используемый для разгрузки кэш-памяти второго уровня. Размер строки кэш-памяти составляет 32 байта. Кроме того, в RT625 имеются логические схемы синхронизации, которые обеспечивают интерфейс между внутренней шиной процессора и SPARC MBus при выполнении асинхронных операций.

RT627 представляет собой статическую память 16К $\times$ 32, специально разработанную для удовлетворения требований hyperSPARC. Она организована как четырехканальная статическая память в виде четырех массивов с логикой побайтной записи и входными и выходными регистрами-защелками. RT627 для ЦП является кэш-памятью с нулевым состоянием ожидания без потерь (т.е. приостановок) на конвейеризацию для всех операций загрузки и записи, которые попадают в кэш-память. RT627 был разработан специально для процессора hyperSPARC, таким образом для соединения с RT620 и RT625 не нужны никакие дополнительные схемы.

Набор кристаллов позволяет использовать преимущества тесной связи процессора с кэш-памятью. Конструкция RT620 допускает потерю одного такта в случае промаха в кэш-памяти первого уровня. Для доступа к кэш-памяти второго уровня в RT620 отведена специальная ступень конвейера. Если происходит промах в кэш-памяти первого уровня, а в кэш-памяти второго уровня имеет место попадание, то центральный процессор не останавливается.

Команды загрузки и записи одновременно генерируют два обращения: одно к кэш-памяти команд первого уровня емкостью 8 Кбайт и другое к кэш-памяти второго уровня. Если адрес команды найден в кэш-памяти первого уровня, то обращение к кэш-памяти второго уровня отменяется и команда становится доступной на стадии декодирования конвейера. Если же во внутренней кэш-памяти произошел промах, а в кэш-памяти второго уровня обнаружено попадание, то команда станет доступной с потерей одного такта, который встроен в конвейер. Такая возможность позволяет конвейеру продолжать непрерывную работу до тех пор, пока имеют место попадания в кэш-память либо первого, либо второго уровня, которые составляют 90% и 98% соответственно для типовых прикладных задач рабочей станции. С целью достижения архитектурного баланса и упрощения обработки исключительных ситуаций целочисленный конвейер и конвейер плавающей точки имеют по пять стадий выполнения операций. Такая конструкция позволяет RT620 обеспечить максимальную пропускную способность, не достижимую в противном случае.

8.2.3. **8.2.3. MicroSPARC-II**

Эффективная с точки зрения стоимости конструкция не может полагаться только на увеличение тактовой частоты. Экономические соображения заставляют принимать решения, основой которых является массовая технология. Системы microSPARC обеспечивают высокую производительность при умеренной тактовой частоте путем оптимизации среднего количества команд, выполняемых за один такт. Это ставит вопросы эффективного управления конвейером и иерархией памяти. Среднее время обращения к памяти должно сокращаться, либо должно возрастать среднее количество команд, выдаваемых для выполнения в каждом такте, увеличивая производительность на основе компромиссов в конструкции процессора.

MicroSPARC-II (рисунок 8.6) является одним из сравнительно недавно появившихся процессоров семейства SPARC. Основное его назначение - однопроцессорные низкостоимостные системы. Он представляет собой высокоинтегрированную микросхему, содержащую целочисленное устройство, устройство управления памятью, устройство плавающей точки, раздельную кэш-память команд и данных, контроллер управления микросхемами динамической памяти и контроллер шины SBus.

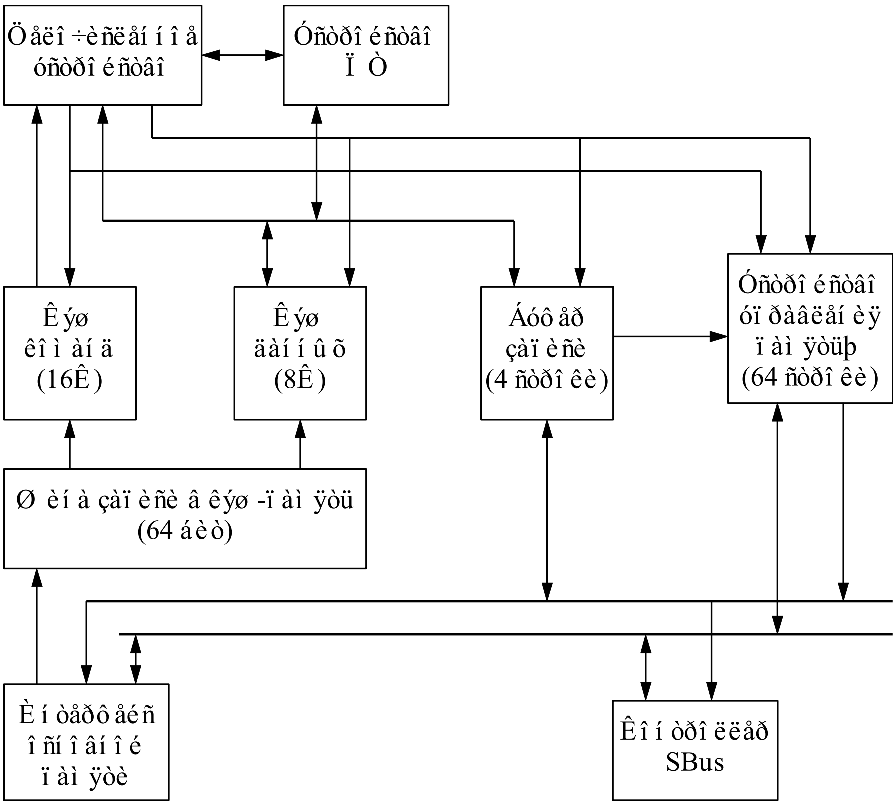


Рис. 8.6. Блок-схема процессора micro Sparc-II

Основными свойствами целочисленного устройства microSPARC-II являются:

- пятиступенчатый конвейер команд;
- предварительная обработка команд переходов;
- поддержка потокового режима работы кэш-памяти команд и данных;
- регистровый файл емкостью 136 регистров (8 регистровых окон);
- интерфейс с устройством плавающей точки;
- предварительная выборка команд с очередью на четыре команды.

Целочисленное устройство использует пятиступенчатый конвейер команд с одновременным запуском до двух команд. Устройство плавающей точки обеспечивает выполнение операций в соответствии со стандартом IEEE 754.

Устройство управления памятью выполняет четыре основных функции. Во-первых, оно обеспечивает формирование и преобразование виртуального адреса в физический. Эта функция реализуется с помощью ассоциативного буфера TLB. Кроме того, устройство управления памятью реализует механизмы защиты памяти. И, наконец, оно выполняет арбитраж обращений к памяти со стороны ввода/вывода, кэша данных, кэша команд и TLB.

Процессор microSPARC II имеет 64-битовую шину данных для связи с памятью и поддерживает оперативную память емкостью до 256 Мбайт. В процессоре интегрирован контроллер шины SBus, обеспечивающий эффективную с точки зрения стоимости реализацию ввода/вывода.

8.2.4. **8.2.4. UltraSPARC**

Основные критерии разработки

Как известно, производительность любого процессора при выполнении заданной программы зависит от трех параметров: такта (или частоты) синхронизации, среднего количества команд, выполняемых за один такт, и общего количества выполняемых в программе команд. Изменить ни один из указанных параметров независимо от других невозможно, поскольку соответствующие базовые технологии взаимосвязаны. Частота синхронизации определяется достигнутым уровнем технологии интегральных схем и функциональной организацией процессора, среднее количество тактов на команду зависит от функциональной организации и архитектуры системы команд, а количество выполняемых в программе команд определяется архитектурой системы команд и технологией компиляторов.

Из сказанного ясно, что создание нового высокопроизводительного процессора требует решения сложных вопросов во всех трех направлениях разработки. При этом эффективная с точки зрения стоимости конструкция не может полагаться только на увеличение тактовой частоты. Экономические соображения заставляют разработчиков принимать решения, основой которых является массовая технология. Системы UltraSPARC-1 обеспечивают высокую производительность при достаточно умеренной тактовой частоте (до 200 МГц) путем оптимизации среднего количества команд, выполняемых за один такт. Однако при таком подходе естественно встают вопросы эффективного управления конвейером команд и иерархией памяти системы. Для увеличения производительности необходимо по возможности уменьшить среднее время доступа к памяти и увеличить среднее количество команд, выдаваемых для выполнения в каждом такте, не превышая при этом разумного уровня сложности процессора.

При разработке суперскалярного процессора практически сразу необходимо "расшить" целый ряд узких мест, ограничивающих выдачу для выполнения нескольких команд в каждом такте. Такими узкими местами являются наличие в программном коде зависимостей по управлению и данным, аппаратные ограничения на количество портов в регистровых файлах процессора и устройствах, реализующих иерархию памяти, а также количество целочисленных конвейеров и конвейеров выполнения операций с плавающей точкой.

При создании своего нового процессора UltraSPARC-1 компания Sun решила добиться увеличения производительности процессора в тех направлениях, где это не противоречило экономическим соображениям. Чтобы сократить число потенциальных проблем, было принято несколько конструкторских решений, которые определили основные характеристики UltraSPARC-1:

- • Реализация на кристалле раздельной кэш-памяти команд и данных
- • Организация широкой выборки команд (128 бит)
- • Создание эффективных средств динамического прогнозирования направления переходов
- • Реализация девятиступенчатого конвейера, обеспечивающего выдачу для выполнения до четырех команд в каждом такте
- • Оптимизация конвейерных операций обращения к памяти

- • Реализация команд обмена данными между памятью и регистрами плавающей точки, позволяющая не приостанавливать диспетчеризацию команд обработки
- • Реализация на кристалле устройства управления памятью (MMU)
- • Расширение набора команд для поддержки графики и обработки изображений
- • Реализация новой архитектуры шины UPA

UltraSPARC-I

Процессор UltraSPARC-1 представляет собой высокопроизводительный, высокоинтегрированной суперскалярный процессор, реализующий 64-битовую архитектуру SPARC-V9. В его состав входят: устройство предварительной выборки и диспетчеризации команд, целочисленное исполнительное устройство, устройство плавающей точки с графическим устройством, устройство управления памятью, устройство загрузки/записи, устройство управления внешней кэш-памятью, устройство управления интерфейсом памяти и кэш-памяти команд и данных (рисунок 8.7).

Рис. 8.7. Блок-схема процессора UltraSPARC-1

Устройство предварительной выборки и диспетчеризации команд

Устройство предварительной выборки и диспетчеризации команд процессора UltraSPARC-1 (PDU) обеспечивает выборку команд в буфер команд, окончательную их дешифрацию, группировку и распределение для параллельного выполнения в конвейерных функциональных устройствах процессора. Буфер команд емкостью в 12 команд позволяет согласовать скорость работы памяти со скоростью обработки исполнительных устройств процессора. Команды могут быть предварительно выбраны из любого уровня иерархии памяти, например, из кэш-памяти команд (I-кэша), внешней кэш-памяти (E-кэша) или из основной памяти системы.

В процессоре реализована схема динамического прогнозирования направления ветвлений программы, основанная на двухбитовой истории переходов и обеспечивающая ускоренную обработку команд условного перехода. Для реализации этой схемы с каждыми двумя командами в I-кэше, связано специальное поле, хранящее двухбитовое значение прогноза. Таким образом, UltraSPARC-1 позволяет хранить информацию о направлении 2048 переходов, что превышает потребности большинства прикладных программ. Поскольку направление перехода может меняться каждый раз, когда обрабатывается соответствующая команда, состояние двух бит прогноза должно каждый раз модифицироваться для отражения реального исхода перехода. Эта схема особенно эффективна при обработке циклов.

Кроме того, в процессоре UltraSPARC-1 с каждыми четырьмя командами в I-кэше связано специальное поле, указывающее на следующую строку кэш-памяти, которая должна выбираться вслед за данной. Использование этого поля позволяет осуществлять выборку командных строк в соответствии с выполняемыми переходами, что обеспечивает для программ с большим числом ветвлений практически ту же самую пропускную способность команд, что и на линейном участке программы. Способность быстро выбрать команды по прогнозируемому целевому адресу команды перехода является очень важной для оптимизации производительности суперскалярного процессора и позволяет UltraSPARC-1 эффективно выполнять "по предположению" (speculative) достаточно хитроумные последовательности условных переходов.

Используемые в UltraSPARC-1 механизмы динамического прогнозирования направления и свертки переходов сравнительно просты в реализации и обеспечивают высокую производительность. По результатам контрольных испытаний UltraSPARC-1 88% переходов по условиям целочисленных операций и 94% переходов по условиям операций с плавающей точкой предсказываются успешно.

Кэш-память команд

Кэш-память команд (I-кэш) представляет собой двухканальную множественно-ассоциативную кэш-память емкостью 16 Кбайт. Она организована в виде 512 строк, содержащих по 32 байта данных. С каждой строкой связан соответствующий адресный тег. Команды, поступающие для записи в I-кэш, проходят предварительное декодирование и записываются в кэш-память вместе с соответствующими признаками, облегчающими их последующую обработку. Окончательное декодирование команд происходит перед их записью в буфер команд.

Организация конвейера

В процессоре UltraSPARC-1 реализован девятиступенчатый конвейер. Это означает, что задержка (время от начала до конца выполнения) большинства команд составляет девять тактов. Однако в любой данный момент времени в процессе обработки могут одновременно находиться до девяти команд, обеспечивая во многих случаях завершение выполнения команд в каждом такте. В действительности эта скорость может быть ниже в связи с природой самих команд, промахами кэш-памяти или другими конфликтами по ресурсам. Первая ступень конвейера - выборка из кэш-памяти команд. На второй ступени команды декодируются и помещаются в буфер команд. Третья ступень, осуществляет группировку и распределение команд по функциональным исполнительным устройствам. В каждом такте на выполнение в исполнительные устройства процессора могут выдаваться по 4 команды (не более двух целочисленных команд или команд плавающей точки/графических команд, одной команды загрузки/записи и одной команды перехода). На следующей ступени происходит выполнение целочисленных команд или вычисляется виртуальный адрес для обращения к памяти, а также осуществляются окончательное декодирование команд плавающей точки (ПТ) и обращение к регистрам ПТ. На пятой ступени происходит обращение к кэш-памяти данных. Определяются попадания и промахи кэш-памяти и разрешаются переходы. При обнаружении промаха кэш-памяти, соответствующая команда загрузки поступает в буфер загрузки. С этого момента целочисленный конвейер ожидает завершения работы конвейеров плавающей точки/графики, которые начинают выполнение соответствующих команд. Затем производится анализ возникновения исключительных ситуаций. На последней ступени все результаты записываются в регистровые файлы и команды изымаются из обработки.

Целочисленное исполнительное устройство

Главной задачей при разработке целочисленного исполнительного устройства (IEU) является обеспечение максимальной производительности при поддержке полной программной совместимости с существующим системным и прикладным ПО. Целочисленное исполнительное устройство UltraSPARC-1 объединяет в себе несколько важных особенностей:

- • 2 АЛУ для выполнения арифметических и логических операций, а также операций сдвига;
- • Многотактные целочисленные устройства умножения и деления;
- • Регистровый файл с восемью окнами и четырьмя наборами глобальных регистров;
- • Реализация цепей ускоренной пересылки результатов;
- • Реализация устройства завершения команд, которое обеспечивает минимальное количество цепей обхода (ускоренной пересылки данных) при построении девятиступенчатого конвейера;
- • Устройство загрузки/записи (LSU).

LSU отвечает за формирование виртуального адреса для всех команд загрузки и записи (включая атомарные операции), за доступ к кэш-памяти данных, а также за буферизацию команд загрузки в случае промаха D-кэша (в буфере загрузки) и буферизацию команд записи (в буфере записи). В каждом такте может выдаваться для выполнения одна команда загрузки и одна команда записи.

Устройство плавающей точки (FPU)

Конвейерное устройство плавающей точки процессора UltraSPARC построено в соответствии со спецификациями архитектуры SPARC-V9 и стандарта IEEE 754. Оно состоит из пяти отдельных функциональных устройств и обеспечивает выполнение операций с плавающей точкой и графических операций. Реализация раздельных исполнительных устройств позволяет UltraSPARC-1 выдавать и выполнять две операции ПТ в каждом такте. Операнды-источники и результаты операций хранятся в регистровом файле емкостью 32 регистра. Большинство команд полностью конвейеризованы (имеют пропускную способность 1 такт), задержку в 3 такта и не зависят от точности операндов (имеют одну и ту же задержку для одинарной и двойной точности). Команды деления и вычисления квадратного корня не конвейеризованы и выполняются 12/22 такта (одинарная/двойная точность), но не останавливают процессор. Другие команды, следующие за командами деления/вычисления квадратного корня, могут выдаваться, выполняться и изыматься из обработки для фиксации результата в регистровом файле до завершения команд деления/вычисления квадратного корня. Процессор поддерживает модель точных прерываний посредством синхронизации конвейера плавающей точки с целочисленным конвейером, а также с помощью средств прогнозирования исключительных ситуаций для операций с большим временем выполнения. FPU может работать с нормализованными и ненормализованными числами с одинарной (32 бит) и двойной точностью (64 бит), а также поддерживает операции над числами с учетверенной точностью (128 бит).

FPU тесно взаимодействует с целочисленным конвейером и способно без каких-либо дополнительных задержек выполнять чтение операнда с ПТ из памяти и следующую за ней операцию ПТ. IEU и FPU имеют выделенный интерфейс управления, который обеспечивает диспетчеризацию операций, выбранных PDU в FPU. Устройство предварительной выборки и диспетчеризации команд выполняет распределение находящихся в очереди команд в FPU. IEU управляет частью операций, связанных с D-кэшем, а FPU выполняет собственно операции обработки данных. При выполнении команд ПТ целочисленное устройство и FPU совместно определяют наличие зависимостей по данным. Существующий между ними интерфейс включает также взаимную синхронизацию при появлении исключительных ситуаций FPU. Для снижения взаимного влияния и увеличения общей производительности в FPU обеспечивается дополнительная буферизация команд ПТ, реализованная с помощью очереди на три команды.

Графическое устройство (GRU)

В процессоре UltraSPARC-1 реализован исчерпывающий набор графических команд, которые обеспечивают аппаратную поддержку высокоскоростной обработки двухмерных и трехмерных изображений, обработку видеоданных и т.д. GRU выполняет операции сложения, сравнения и логические операции над 16-битовыми и 32-битовыми целыми числами, а также операции умножения над 8-битовыми и 16-битовыми целыми. В GRU поддерживаются однотактные операции определения расстояния между пикселами, операции выравнивания данных, операции упаковки и слияния.

Устройство управления памятью (MMU)

Высокая суперскалярная производительность процессора поддерживается высокой скоростью поступления для обработки команд и данных. Обычно эта задача ложится на иерархию памяти системы. Устройство управления памятью процессора UltraSPARC-1 выполняет все операции обращения к памяти, реализуя необходимые средства поддержки виртуальной памяти. Виртуальное адресное пространство задачи определяется 64-битовым виртуальным адресом, однако процессор UltraSPARC-1 поддерживает только 44-битовое виртуальное адресное пространство. Соответствующее преобразование является функцией операционной системы.

В свою очередь MMU обеспечивает отображение 44-битового виртуального адреса в 41-битовый физический адрес памяти. Это преобразование выполняется с помощью полностью ассоциативных 64-строчных буферов: iTLB - для команд и dTLB - для данных. Каждый из этих буферов по существу представляет собой полностью ассоциативную кэш-память дескрипторов страниц. В каждой строке TLB хранится информация о виртуальном адресе страницы, соответствующем физическом адресе страницы, а также о допустимом режиме доступа к странице и ее использовании. Процесс преобразования виртуального адреса в физический заканчивается сразу, если при поиске в кэш-памяти TLB происходит попадание (соответствующая строка находится в TLB). В противном случае замещение строки TLB осуществляется специальным аппаратно-программным механизмом. MMU поддерживает четыре размера страниц: 8K, 64K, 512K и 4Мбайт.

Как уже было отмечено, MMU реализует также механизмы защиты и контроля доступа к памяти. В результате выполняющийся процесс не может иметь доступ к адресному пространству других процессов, и кроме того, гарантируется заданный режим доступа процесса к определенным областям памяти (на базе информации о допустимом режиме доступа к страницам памяти). Например, процесс не может модифицировать страницы памяти, доступ к которым разрешен только по чтению, или которые зарезервированы для размещения системных программ и т.д.

Наконец, MMU выполняет функции определения порядка (приоритет) обращений к памяти со стороны ввода/вывода, D-кэша, I-кэша и схем преобразования виртуального адреса в физический.

Управление интерфейсом памяти (MIU)

В процессоре UltraSPARC-1 применяется специальная подсистема ввода/вывода (MIU), которая обеспечивает управление всеми операциями ввода и вывода, которые осуществляются между локальными ресурсами: процессором, основной памятью, схемами управления и всеми внешними ресурсами системы. В частности, все системные транзакции, связанные с обработкой промахов кэш-памяти, прерываниями, наблюдением за когерентным состоянием кэш-памяти, операциями обратной записи и т.д., обрабатываются MIU. MIU взаимодействует с системой на частоте меньшей, чем частота UltraSPARC-1 в соотношении 1/2, или 1/3.

Кэш-память данных (D-кэш)

В процессоре UltraSPARC-1 используется кэш-память данных с прямым отображением емкостью 16 Кбайт, реализующая алгоритм сквозной записи. D-кэш организован в виде 512 строк, в каждой строке размещаются два 16-байтных подблока данных. С каждой строкой связан соответствующий адресный тег. D-кэш индексируется с помощью виртуального адреса, при этом теги также хранят соответствующую часть виртуального адреса. При возникновении промаха при обращении к кэшируемой ячейке памяти происходит загрузка 16-байтного подблока из основной памяти. Поиск слова в D-кэше осуществляется с помощью виртуального адреса. Младшие разряды этого адреса обеспечивают доступ к строке кэш-памяти, содержащей требуемое слово (прямое отображение). Старшие разряды виртуального адреса сравниваются затем с битами соответствующего тега для определения попадания или промаха. Подобная схема гарантирует быстрое обнаружение промаха и обеспечивает преобразование виртуального адреса в физический только при наличии промаха.

Управление внешней кэш-памятью (E-кэшем)

Одной из важнейших проблем построения системы является согласование производительности процессора со скоростью основной памяти. Основными методами решения этой проблемы (помимо различных способов организации основной памяти и системы межсоединений) являются увеличение размеров и многоуровневая организация кэш-памяти. Устройство управления внешней кэш-памятью (ECU) процессора UltraSPARC-1 позволяет эффективно обрабатывать промахи кэш-памяти данных (D-кэша) и команд (E-кэша). Все обращения к внешней кэш-памяти (E-кэшу) конвейеризованы, выполняются за 3 такта и осуществляют пересылку 16 байт команд или данных в каждом такте. Такая организация дает возможность эффективно планировать конвейерное выполнение программного кода, содержащего большой объем обрабатываемых данных, и минимизировать потери производительности, связанные с обработкой промахов в D-кэше. ECU позволяет наращивать объем внешней кэш-памяти от 512 Кбайт до 4 Мбайт.

ECU обеспечивает совмещенную во времени обработку промахов обращений по чтению данных из E-кэша с операциями записи. Например, во время обработки промаха по загрузке ECU разрешает поступление запросов по записи данных в E-кэш. Кроме того, ECU поддерживает операции наблюдения (snoops), связанные с обеспечением когерентного состояния памяти системы.

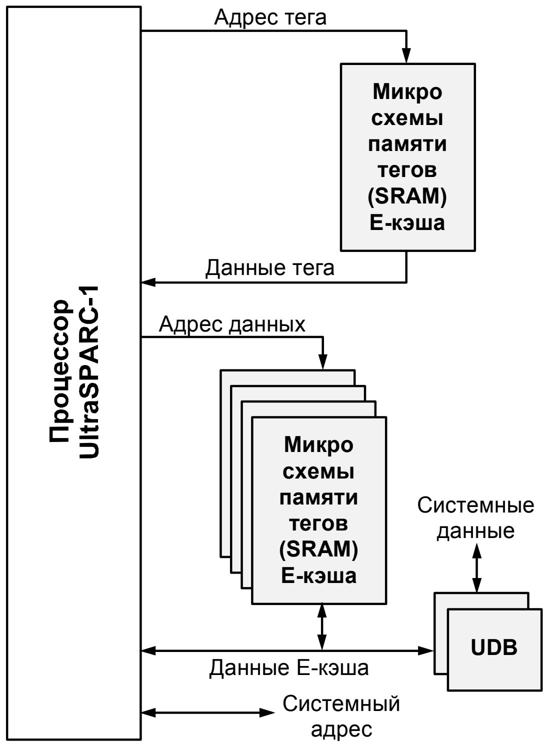


Рис.8.8. Типовой процессорный модуль

Типовой процессорный модуль UltraSPARC-1

Типовой процессорный модуль (рисунок 8.8). UltraSPARC-1 состоит из собственно процессора UltraSPARC-1, микросхем синхронной статической памяти (SRAM), используемых для построения памяти тегов и данных внешнего кэша и двух кристаллов буферов системных данных (UDB). UDB изолируют внешний кэш процессора от остальной части системы и обеспечивают буферизацию данных для приходящих и исходящих системных транзакций, а также формирование, проверку контрольных разрядов и автоматическую коррекцию данных (с помощью ECC-кодов). Таким образом, UDB позволяет интерфейсу работать на тактовой частоте процессора (за счет снижения емкостной нагрузки).

Буфер данных обеспечивает также совмещение во времени системных транзакций с локальными транзакциями Е-кэша. В состав процессора UltraSPARC-1 включена логика управления буферными кристаллами, которая обеспечивает быструю пересылку данных между процессором или внешним кэшем и системой. Для поддержки системных транзакций используется отдельная адресная шина и отдельный набор управляющих сигналов.

Архитектура системной шины UPA

Высокая производительность процессора UltraSPARC-1 потребовала создания гибкой масштабируемой архитектуры межсоединений, позволяющей достаточно просто строить системы для широкого круга приложений от небольших настольных систем индивидуального пользования до больших многопроцессорных серверов масштаба предприятия. Новая архитектура UPA (Ultra Port Architecture) определяет возможности построения целого семейства тесно связанных многопроцессорных систем с общей памятью.

UPA представляет собой спецификацию, описывающую логические и физические интерфейсы порта системной шины и требования, накладываемые на организацию

межсоединений. К этим портам подключаются все устройства системы. Спецификация UPA включает также описание поведения системного контроллера и интерфейс ввода/вывода системы межсоединений.
UPA может поддерживать большое количество (рисунок 8.9) системных портов (32, 64, 128 и т.д.) и включает четыре типа интерфейса. Интерфейс главного устройства выдает в систему межсоединений транзакции чтения/записи по физическому адресу, используя распределенный протокол арбитража для управления адресной шиной. Главное устройство UPA (например, процессорный модуль UltraSPARC-1) может включать физически адресуемую когерентную кэш-память, на размер которой в общем случае не накладывается никаких ограничений. Интерфейс подчиненного устройства получает транзакции чтения/записи от главных устройств UPA, поддерживая строгое упорядочивание транзакций одного и того же класса главных устройств, а также транзакций, направляемых по одному и тому же адресу устройства. Порт UPA может быть только подчиненным, например, для подключения графического буфера кадров. Двумя другими дополнительными интерфейсами порта UPA являются источник прерывания и обработчик прерываний. Источники прерывания UPA генерируют пакеты прерывания, направляемые к обработчикам прерываний UPA.

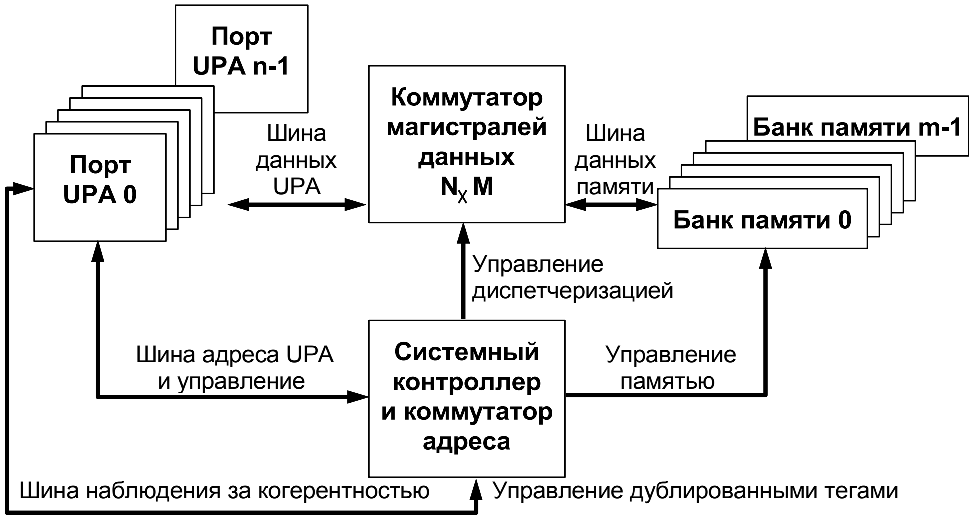


Рис. 8.9. Масштабируемая архитекутра UPA

В отличие от традиционных мультипроцессорных систем, которые поддерживают когерентное состояние кэш-памяти и разделяют глобально наблюдаемую адресную шину, архитектура межсоединений UPA основана на пакетной коммутации сообщений по принципу точка-точка. Поддержка когерентного состояния кэш-памяти системы для настольных рабочих станций, включающих от 1 до 4 процессоров, осуществляется централизованным системным контроллером, а для больших серверов - распределенным системным контроллером. UPA может поддерживать дублирование наборов тегов всех кэшей системы и позволяет для каждой когерентной транзакции выполнять параллельно просмотр дублированных тегов и обращение к основной памяти.
Отход от традиционных методов построения мультипроцессорных систем, основанных на наблюдаемой шине или на справочнике, позволяет существенно минимизировать задержки доступа к данным благодаря сокращению потерь на обработку промахов кэш-памяти. В итоге архитектура межсоединений UPA позволяет полностью использовать высокую пропускную способность процессора UltraSPARC-1. Максимальная скорость передачи данных составляет 1.3 Гбайт/с при работе UPA на тактовой частоте 83 МГц.

Разработчики архитектуры UPA многое сделали с целью минимизации задержек доступа к данным. Например, UPA поддерживает раздельные шины адреса и данных. Именно эти широкие шины (адресная шина имеет ширину 64 бит (в соответствии со спецификацией 64-битовой архитектуры V9), а шина данных - 144 бит (128 бит данных и 16 бит для контроля ошибок)) обеспечивают пиковую пропускную способность системы. Наличие отдельных шин позволяет устранить задержки, возникающие при переключении разделяемой шины между данными и адресом, а также возможные конфликты доступа к общей шине.
UPA не только поддерживает отдельные шины адреса и данных, но позволяет также иметь несколько шин с организацией соединений точка-точка. Обычно в большинстве систем имеются несколько интерфейсов для обеспечения работы подсистемы ввода/вывода, графической подсистемы и процессора. В мультипроцессорных системах требуются также дополнительные интерфейсы для организации связи между несколькими ЦП. Вместо одного набора шин данных и адреса для всех этих интерфейсов UPA допускает создание неограниченного количества шин.
Подобная организация имеет ряд достоинств. Наличие нескольких наборов шин позволяет минимизировать количество циклов арбитража и уменьшает вероятность конфликтов. Системный контроллер несет ответственность за работу и взаимодействие различных шин и может параллельно обрабатывать запросы нескольких шин. Он позволяет также уменьшить задержки, связанные с захватом шины. По существу, наличие нескольких шин адреса и данных означает меньшее число потенциальных главных устройств на каждом наборе шин. Для обеспечения наименьшей возможной задержки захвата шины используется распределенный конвейеризованный протокол арбитража. Каждый порт UPA имеет собственные схемы арбитража, при этом каждый порт в системе видит запросы шины всех других портов. Такая схема также позволяет уменьшить задержку доступа и обеспечивает увеличение общей производительности системы.
Архитектура UPA легко адаптируется для работы почти с любой конфигурацией системы (от однопроцессорной до массивно-параллельной). Разработчиками были предприняты специальные усилия с целью ее оптимизации для систем, содержащих от 1 до 4 процессоров. В результате до четырех тесно связанных процессоров и системный контроллер могут разделять доступ к одной и той же системной адресной шине. Однако на базе богатого набора транзакций и протокола когерентности, которые поддерживаются устройством интерфейса памяти процессора UltraSPARC-1 могут быть построены мультипроцессорные системы с большим количеством процессоров. В архитектуре UPA применяется протокол когерентности, построенный на основе операций записи с аннулированием соответствующих копий блока в кэш-памяти других процессоров системы и использующий для наблюдения дублированные теги. Процессор UltraSPARC поддерживает переходы состояний блоков кэш-памяти, соответствующие протоколам MOESI, MOSI и MSI.
Следует отметить, что в основу архитектуры UPA положены настолько гибкие принципы, что она позволяет иметь в системе не только несколько шин (мультиплексированных или раздельных), но и в широких пределах варьировать разрядность шины данных для удовлетворения различных требований к отношению стоимость / производительность. При этом в различных частях системы в зависимости от конкретных требований может использоваться разная скорость передачи данных. Например, разрядность шины данных системы ввода/вывода вполне может быть ограничена 64 битами, но для согласования с интерфейсом процессора более предпочтительна разрядность в 128 бит. С другой стороны, разрядность шины данных оперативной памяти системы может быть еще более увеличена для обеспечения высокой пропускной способности при использовании более медленных, но более дешевых микросхем памяти (в младших моделях компьютеров на базе

микропроцессора UltraSPARC-1 используется 256-битовая шина данных памяти, а в старших моделях - 512-битовая).

Набор графических команд

UltraSPARC является первым универсальным процессором с 64-битовой архитектурой, обеспечивающим высокую пропускную способность, необходимую для реализации высокоскоростной графики и обработки видеоизображений в реальном масштабе времени. Расширенный набор команд UltraSPARC позволяет быстро (за один такт) выполнять достаточно сложные графические операции, для реализации которых обычно затрачивается несколько десятков тактов. При этом только три процента реальной площади кристалла было потрачено для реализации графических команд. Высокая производительность UltraSPARC и его способность выполнять декомпрессию и обработку видеоданных в реальном времени позволяют в ряде случае при построении системы обойтись без специальных дорогостоящих видеопроцессоров.

Высокоскоростная обработка графики и видеоизображений базируется на суперскалярной архитектуре процессора UltraSPARC. При этом для адресации данных (вычисления адресов команд загрузки и записи) широко используются целочисленные регистры, а для манипуляций с данными - регистры плавающей точки. Такое функциональное разделение регистров существенно увеличивает пропускную способность процессора, обеспечивая приложению максимальное количество доступных регистров и параллельное выполнение команд.

Специальный набор видеокоманд UltraSPARC (VIS - Video Instruction Set) предоставляет широкие возможности обработки графических данных: команды упаковки и распаковки пикселей, команды параллельного сложения, умножения и сравнения данных, представленных в нескольких целочисленных форматах, команды выравнивания и слияния, обработки контуров изображений и адресации массивов. Эти графические команды оптимизированы для работы с малоразрядной целочисленной арифметикой, при использовании которой обычно возникают значительные накладные расходы из-за необходимости частого преобразования целочисленного формата в формат ПТ и обратно. Возможность увеличения разрядности промежуточных результатов обеспечивает дополнительную точность, необходимую для высококачественных графических изображений. Все операнды графических команд находятся в регистрах ПТ, что обеспечивает максимальное количество регистров для хранения промежуточных результатов вычислений и параллельное выполнение команд.

UltraSPARC поддерживает различные алгоритмы компрессии, используемые для разнообразных видеоприложений и обработки неподвижных изображений, включая H.261, MPEG-1, MPEG-2 и JPEG. Более того, он может обеспечивать скорости кодирования и декодирования, необходимые для организации видеоконференций в реальном времени.

Первые системы на базе нового процессора

В настоящее время Sun выпускает два типа настольных рабочих станций и серверов, оснащенных процессорами UltraSPARC: Ultra 1 и Ultra 2 (рисунок 8.10). В моделях Ultra 1 используются процессоры с тактовой частотой 143 и 167 МГц. При этом они комплектуются как стандартными видеоадаптерами TurboGX и TurboGXplus (модели 140 и 170), так и новыми мощными видеоподсистемами Creator и Creator3D (модель 170E), позволяют наращивать объем оперативной памяти до 512 Мбайт, внутренних дисков до 4.2 Гбайт и устанавливать накопители на магнитной ленте, флоппи-дисководы и считывающие устройства с компакт-дисков. Эти системы обеспечивают уровень производительности в 252 SPECint92 и 351 SPECfp92 при тактовой частоте 167 МГц. Модели 170E оснащаются контроллерами Fast&Wide SCSI-2 и 100Base-T Ethernet.

Модели Ultra 2 - это однопроцессорные и двухпроцессорные системы на базе 200 МГц процессора UltraSPARC (332 SPECint92 и 505 SPECfp92), имеющие максимальный объем оперативной памяти 1 Гбайт. Появление следующих моделей, построенных на процессорах UltraSPARC II (420 SPECint92 и 660 SPECfp92), ожидается в середине 1996 года.

Таким образом, выпуск 64-битового процессора UltraSPARC и первых компьютеров на его основе ознаменовал собой новый этап в развитии Sun Microsystems. Компания планирует постепенно перевести на эти процессоры все свои изделия, включая рабочие станции и серверы начального уровня. Конечно для широкого внедрения новой концепции обработки данных, получившей название UltraComputing, понадобится некоторое время, но уже сейчас очевидно, что ориентация Sun на обеспечение высокой сбалансированной производительности для широкого класса прикладных систем, высокой пропускной способности передачи данных для сетевых приложений и построение эффективных средств визуализации и обработки видеоданных в реальном времени позволяет ей сохранять лидирующие позиции на современном рынке компьютеров для научно-технических и бизнес-приложений.

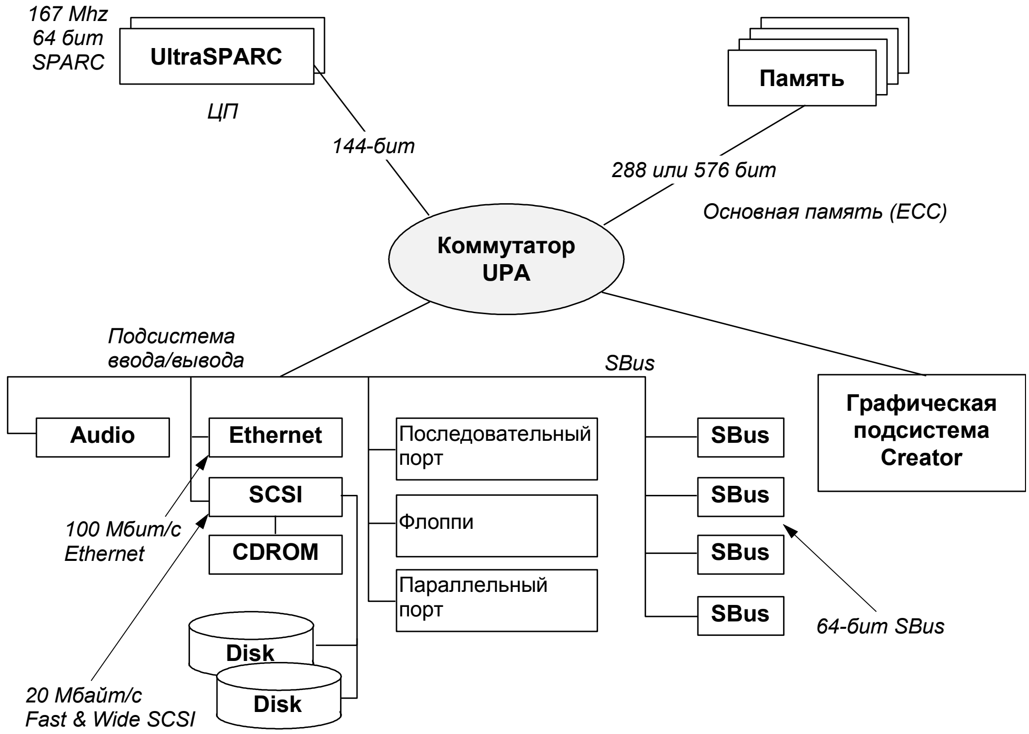


Рис. 8.10. Архитектура компьютеров Ultra 1 и Ultra 2

8.3. 8.3. Процессоры PA-RISC компании Hewlett-Packard

Основой разработки современных изделий Hewlett-Packard является архитектура PA-RISC. Она была разработана компанией в 1986 году и с тех пор прошла несколько стадий своего развития благодаря успехам интегральной технологии от многокристального до однокристального исполнения. В сентябре 1992 года компания Hewlett-Packard объявила о создании своего суперскалярного процессора PA-7100, который с тех пор стал основой построения семейства рабочих станций HP 9000 Series 700 и семейства бизнес-серверов HP 9000 Series 800. В настоящее время имеются 33-, 50- и 99 МГц реализации кристалла PA-7100. Кроме того, выпущены модифицированные, улучшенные по многим параметрам кристаллы PA-7100LC с тактовой частотой 64, 80 и 100 МГц, и PA-7150 с тактовой частотой 125 МГц, а также PA-7200 с тактовой частотой 90 и 100 МГц. Компания активно разрабатывает процессор следующего поколения HP 8000, которые будет работать с тактовой частотой 200 МГц и обеспечивать уровень 360 единиц SPECint92 и 550 единиц SPECfp92. Появление этого кристалла ожидается в 1996 году. Кроме того, Hewlett-Packard в сотрудничестве с Intel планируют создать новый процессор с очень длинным командным словом (VLIW-архитектура), который будет совместим как с семейством Intel x86, так и семейством PA-RISC. Выпуск этого процессора планировалась на 1998 год.

PA 7100

Особенностью архитектуры PA-RISC является внекристальная реализация кэша, что позволяет реализовать различные объемы кэш-памяти и оптимизировать конструкцию в зависимости от условий применения (рисунок 8.11). Хранение команд и данных осуществляется в раздельных кэшах, причем процессор соединяется с ними с помощью высокоскоростных 64-битовых шин. Кэш-память реализуется на высокоскоростных кристаллах статической памяти (SRAM), синхронизация которых осуществляется непосредственно на тактовой частоте процессора. При тактовой частоте 100 МГц каждый кэш имеет полосу пропускания 800 Мбайт/с при выполнении операций считывания и 400 Мбайт/с при выполнении операций записи. Микропроцессор аппаратно поддерживает различный объем кэш-памяти: кэш команд может иметь объем от 4 Кбайт до 1 Мбайт, кэш данных - от 4 Кбайт до 2 Мбайт. Чтобы снизить коэффициент промахов применяется механизм хеширования адреса. В обоих кэшах для повышения надежности применяются дополнительные контрольные разряды, причем ошибки кэша команд корректируются аппаратными средствами.

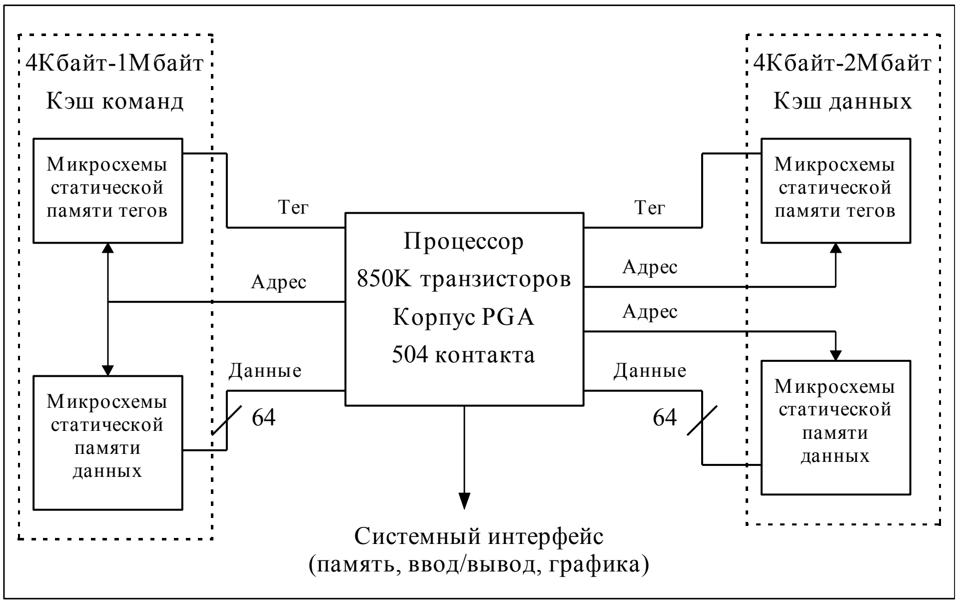


Рис. 8.11. Блок-схема процессора PA 7100

Процессор подсоединяется к памяти и подсистеме ввода/вывода посредством синхронной шины. Процессор может работать с тремя разными отношениями внутренней и внешней тактовой частоты в зависимости от частоты внешней шины: 1:1, 3:2 и 2:1. Это позволяет использовать в системах разные по скорости микросхемы памяти.

Конструктивно на кристалле PA-7100 размещены целочисленный процессор, процессор для обработки чисел с плавающей точкой, устройство управления кэшем, унифицированный буфер TLB, устройство управления, а также ряд интерфейсных схем. Целочисленный процессор включает АЛУ, устройство сдвига, сумматор команд перехода, схемы проверки кодов условий, схемы обхода, универсальный регистровый файл, регистры управления и регистры адресного конвейера. Устройство управления кэш-памятью содержит регистры, обеспечивающие перезагрузку кэш-памяти при возникновении промахов и контроль когерентного состояния памяти. Это устройство содержит также адресные регистры сегментов, буфер преобразования адреса TLB и аппаратуру хеширования, управляющую перезагрузкой TLB. В состав процессора плавающей точки входят устройство умножения, арифметико-логическое устройство, устройство деления и извлечения квадратного корня, регистровый файл и схемы "закоротки" результата. Интерфейсные устройства включают все необходимые схемы для связи с кэш-памятью команд и данных, а также с шиной данных. Обобщенный буфер TLB содержит 120 строк ассоциативной памяти фиксированного размера и 16 строк переменного размера.

Устройство плавающей точки (рисунок 8.12) реализует арифметику с одинарной и двойной точностью в стандарте IEEE 754. Его устройство умножения используется также для выполнения операций целочисленного умножения. Устройства деления и вычисления квадратного корня работают с удвоенной частотой процессора. Арифметико-логическое устройство выполняет операции сложения, вычитания и преобразования форматов данных. Регистровый файл состоит из 28 64-битовых регистров, каждый из которых может использоваться как два 32-битовых регистра для выполнения операций с плавающей точкой одинарной точности. Регистровый файл имеет пять портов чтения и три порта записи, которые обеспечивают одновременное выполнение операций умножения, сложения и загрузки/записи.

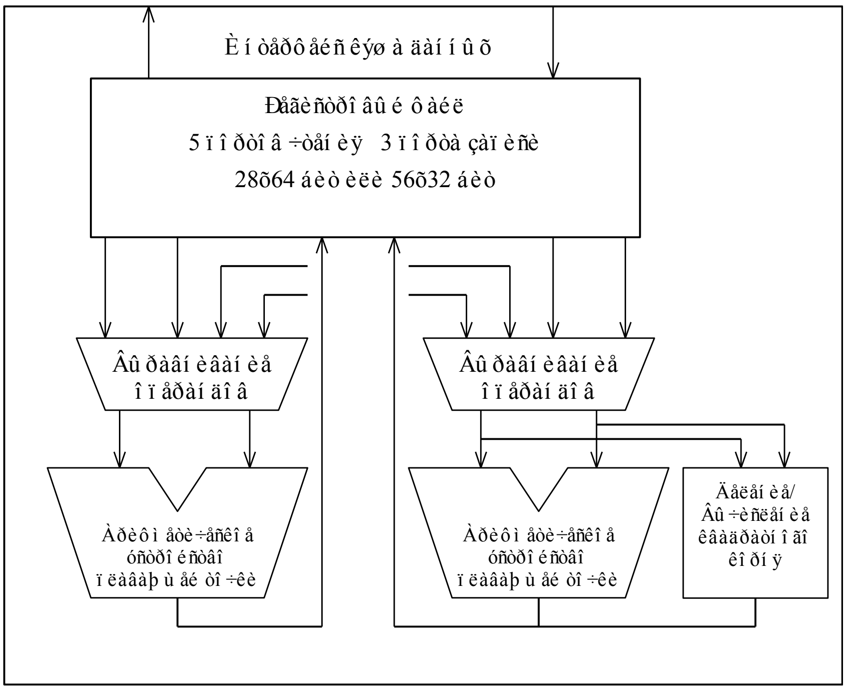


Рис. 8.12. Управление командами плавающей точки

Большинство улучшений производительности процессора связано с увеличением тактовой частоты до 100 МГц по сравнению с 66 МГц у его предшественника.

Конвейер целочисленного устройства включает шесть ступеней: Чтение из кэша команд (IR), Чтение операндов (OR), Выполнение/Чтение из кэша данных (DR), Завершение чтения кэша данных (DRC), Запись в регистры (RW) и Запись в кэш данных (DW). На ступени ID выполняется выборка команд. Реализация механизма выдачи двух команд требует небольшого буфера предварительной выборки, который

обеспечивает предварительную выборку команд за два такта до начала работы ступени IR. Во время выполнения на ступени OR все исполнительные устройства декодируют поля операндов в команде и начинают вычислять результат операции. На ступени DR целочисленное устройство завершает свою работу. Кроме того, кэш-память данных выполняет чтение, но данные не поступают до момента завершения работы ступени DRC. Результаты операций сложения (ADD) и умножения (MULTIPLY) также становятся достоверными в конце ступени DRC. Запись в универсальные регистры и регистры плавающей точки производится на ступени RW. Запись в кэш данных командами записи (STORE) требует двух тактов. Наиболее раннее двухтактное окно команды STORE возникает на ступенях RW и DW. Однако это окно может сдвигаться, поскольку записи в кэш данных происходят только когда появляется следующая команда записи. Операции деления и вычисления квадратного корня для чисел с плавающей точкой заканчиваются на много тактов позже ступени DW.
Конвейер проектировался с целью максимального увеличения времени, необходимого для выполнения чтения внешних кристаллов SRAM кэш-памяти данных. Это позволяет максимизировать частоту процессора при заданной скорости SRAM. Все команды загрузки (LOAD) выполняются за один такт и требуют только одного такта полосы пропускания кэш-памяти данных. Поскольку кэши команд и данных размещены на разных шинах, в конвейере отсутствуют какие-либо потери, связанные с конфликтами по обращениям в кэш данных и кэш команд.
Процессор может в каждом такте выдавать на выполнение одну целочисленную команду и одну команду плавающей точки. Полоса пропускания кэша команд достаточна для поддержания непрерывной выдачи двух команд в каждом такте. Отсутствуют какие-либо ограничения по выравниванию или порядку следования пары команд, которые выполняются вместе. Кроме того, отсутствуют потери тактов, связанных с переключением с выполнения двух команд на выполнение одной команды. Специальное внимание было уделено тому, чтобы выдача двух команд в одном такте не приводила к ограничению тактовой частоты. Чтобы добиться этого, в кэше команд был реализован специально предназначенный для этого заранее декодируемый бит, чтобы отделить команды целочисленного устройства от команд устройства плавающей точки. Этот бит предварительного декодирования команд минимизирует время, необходимое для правильного разделения команд.
Потери, связанные с зависимостями по данным и управлению, в этом конвейере минимальны. Команды загрузки выполняются за один такт, за исключением случая, когда последующая команда пользуется регистром-приемником команды LOAD. Как правило компилятор позволяет обойти подобные потери одного такта. Для уменьшения потерь, связанных с командами условного перехода, в процессоре используется алгоритм прогнозирования направления передачи управления. Для оптимизации производительности циклов передачи управления вперед по программе прогнозируются как невыполняемые переходы, а передачи управления назад по программе - как выполняемые переходы. Правильно спрогнозированные условные переходы выполняются за один такт.
Количество тактов, необходимое для записи слова или двойного слова командой STORE уменьшено с трех до двух тактов. В более ранних реализациях архитектуры PA-RISC был необходим один дополнительный такт для чтения тега кэша, чтобы гарантировать попадание, а также для того, чтобы объединить старые данные строки кэш-памяти данных с записываемыми данными. PA 7100 использует отдельную шину адресного тега, чтобы совместить по времени чтение тега с записью данных предыдущей команды STORE. Кроме того, наличие отдельных сигналов разрешения записи для каждого слова строки кэш-памяти устраняет необходимость объединения старых данных с новыми, поступающими при выполнении команд записи слова или двойного слова. Этот алгоритм требует, чтобы запись в микросхемы SRAM происходила только после того, когда будет определено, что данная запись

сопровождается попаданием в кэш и не вызывает прерывания. Это требует дополнительной ступени конвейера между чтением тега и записью данных. Такая конвейеризация не приводит к дополнительным потерям тактов, поскольку в процессоре реализованы специальные цепи обхода, позволяющие направить отложенные данные команды записи последующим командам загрузки или командам STORE, записывающим только часть слова. Для данного процессора потери конвейера для команд записи слова или двойного слова сведены к нулю, если непосредственно последующая команда не является командой загрузки или записи. В противном случае потери равны одному такту. Потери на запись части слова могут составлять от нуля до двух тактов. Моделирование показывает, что подавляющее большинство команд записи в действительности работают с однословным или двухсловным форматом.
Все операции с плавающей точкой, за исключением команд деления и вычисления квадратного корня, полностью конвейеризованы и имеют двухтактную задержку выполнения как в режиме с одинарной, так и с двойной точностью. Процессор может выдавать на выполнение независимые команды с плавающей точкой в каждом такте при отсутствии каких-либо потерь. Последовательные операции с зависимостями по регистрам приводят к потере одного такта. Команды деления и вычисления квадратного корня выполняются за 8 тактов при одиночной и за 15 тактов при двойной точности. Выполнение команд не останавливается из-за команд деления/вычисления квадратного корня до тех пор, пока не потребуется регистр результата или не будет выдаваться следующая команда деления/вычисления квадратного корня.
Процессор может выполнять параллельно одну целочисленную команду и одну команду с плавающей точкой. При этом "целочисленными командами" считаются и команды загрузки и записи регистров плавающей точки, а "команды плавающей точки" включают команды FMPYADD и FMPYSUB. Эти последние команды объединяют операцию умножения с операциями сложения или вычитания соответственно, которые выполняются параллельно. Пиковая производительность составляет 200 MFLOPS для последовательности команд FMPYADD, в которых смежные команды независимы по регистрам.
Потери для операций плавающей точки, использующих предварительную загрузку операнда командой LOAD, составляют один такт, если команды загрузки и плавающей арифметики являются смежными, и два такта, если они выдаются для выполнения одновременно. Для команды записи, использующей результат операции с плавающей точкой, потери отсутствуют, даже если они выполняются параллельно.
Потери, возникающие при промахах в кэше данных, минимизируются посредством применения четырех разных методов: "попадание при промахе" для команд LOAD и STORE, потоковый режим работы с кэшем данных, специальная кодировка команд записи, позволяющая избежать копирования строки, в которой произошел промах, и семафорные операции в кэш-памяти. Первое свойство позволяет во время обработки промаха в кэше данных выполнять любые типы других команд. Для промахов, возникающих при выполнении команды LOAD, обработка последующих команд может продолжаться до тех пор, пока регистр результата команды LOAD не потребуется в качестве регистра операнда для другой команды. Компилятор может использовать это свойство для предварительной выборки в кэш необходимых данных задолго до того момента, когда они действительно потребуются. Для промахов, возникающих при выполнении команды STORE, обработка последующих команд загрузки или операций записи в части одного слова продолжается до тех пор, пока не возникает обращений к строке, в которой произошел промах. Компилятор может использовать это свойство для выполнения команд на фоне записи результатов предыдущих вычислений. Во время задержки, связанной с обработкой промаха, другие команды LOAD и STORE, для которых происходит попадание в кэш данных, могут выполняться как и другие команды целочисленной арифметики и плавающей точки. В течение всего времени обработки промаха команды STORE, другие команды записи в

ту же строку кэш-памяти могут происходить без дополнительных потерь времени. Для каждого слова в строке кэш-памяти процессор имеет специальный индикационный бит, предотвращающий копирование из памяти тех слов строки, которые были записаны командами STORE. Эта возможность применяется к целочисленным и плавающим операциям LOAD и STORE.

Выполнение команд останавливается, когда регистр-приемник команды LOAD, выполняющейся с промахом, требуется в качестве операнда другой команды. Свойство "потоковости" позволяет продолжить выполнение как только нужное слово или двойное слово возвращается из памяти. Таким образом, выполнение команд может продолжаться как во время задержки, связанной с обработкой промаха, так и во время заполнения соответствующей строки при промахе.

При выполнении блочного копирования данных в ряде случаев компилятор заранее знает, что запись должна осуществляться в полную строку кэш-памяти. Для оптимизации обработки таких ситуаций архитектура PA-RISC 1.1 определяет специальную кодировку команд записи ("блочное копирование"), которая показывает, что аппаратуре не нужно осуществлять выборку из памяти строки, при обращении к которой может произойти промах кэш-памяти. В этом случае время обращения к кэшу данных складывается из времени, которое требуется для копирования в память старой строки кэш-памяти по тому же адресу в кэше (если он "грязный") и времени, необходимого для записи нового тега кэша. В процессоре PA 7100 такая возможность реализована как для привилегированных, так и для непривилегированных команд.

Последнее улучшение управления кэшем данных связано с реализацией семафорных операций "загрузки с обнулением" непосредственно в кэш-памяти. Если семафорная операция выполняется в кэше, то потери времени при ее выполнении не превышают потерь обычных операций записи. Это не только сокращает конвейерные потери, но и снижает трафик шины памяти. В архитектуре PA-RISC 1.1 предусмотрен также другой тип специального кодирования команд, который устраняет требование синхронизации семафорных операций с устройствами ввода/вывода.

Управление кэш-памятью команд позволяет при промахе продолжить выполнение команд сразу же после поступления отсутствующей в кэше команды из памяти. 64-битовая магистраль данных, используемая для заполнения блоков кэша команд, соответствует максимальной полосе пропускания внешней шины памяти 400 Мбайт/с при тактовой частоте 100 МГц.

В процессоре предусмотрен также ряд мер по минимизации потерь, связанных с преобразованиями виртуальных адресов в физические.

Конструкция процессора обеспечивает реализацию двух способов построения многопроцессорных систем. При первом способе каждый процессор подсоединяется к интерфейсному кристаллу, который наблюдает за всеми транзакциями на шине основной памяти. В такой системе все функции по поддержанию когерентного состояния кэш-памяти возложены на интерфейсный кристалл, который посылает процессору соответствующие транзакции. Кэш данных построен на принципах отложенного обратного копирования и для каждого блока кэш-памяти поддерживаются биты состояния "частный" (private), "грязный" (dirty) и "достоверный" (valid), значения которых меняются в соответствии с транзакциями, которые выдает или принимает процессор.

Второй способ организации многопроцессорной системы позволяет объединить два процессора и контроллер памяти и ввода-вывода на одной и той же локальной шине памяти. В такой конфигурации не требуется дополнительных интерфейсных кристаллов и она совместима с существующей системой памяти. Когерентность кэш-памяти обеспечивается наблюдением за локальной шиной памяти. Пересылки строк между кэшами выполняются без участия контроллера памяти и ввода-вывода. Такая конфигурация обеспечивает возможность построения очень дешевых высокопроизводительных многопроцессорных систем.

Процессор поддерживает ряд операций, необходимых для улучшения графической производительности рабочих станций серии 700: блочные пересылки, Z-буфери-зацию, интерполяцию цветов и команды пересылки данных с плавающей точкой для обмена с пространством ввода/вывода.
Процессор построен на базе технологического процесса КМОП с проектными нормами 0.8 микрон, что обеспечивает тактовую частоту 100 МГц.
PA 7200
Процессор PA 7200 имеет ряд архитектурных усовершенствований по сравнению с PA 7100, главными из которых являются добавление второго целочисленного конвейера, построение внутрикристального вспомогательного кэша данных и реализация нового 64-битового интерфейса с шиной памяти.
Процессор PA 7200, как и его предшественник, обеспечивает суперскалярный режим работы с одновременной выдачей до двух команд в одном такте. Все команды процессора можно разделить на три группы: целочисленные операции, операции загрузки/записи и операции с плавающей точкой. PA 7200 осуществляет одновременную выдачу двух команд, принадлежащим разным группам, или двух целочисленных команд (благодаря наличию второго целочисленного конвейера с АЛУ и дополнительных портов чтения и записи в регистровом файле). Команды перехода выполняются в целочисленном конвейере, причем эти переходы могут составлять пару для одновременной выдачи на выполнение только с предшествующей командой.
Повышение тактовой частоты процессора требует упрощения декодирования команд на этапе выдачи. С этой целью предварительная дешифрация потока команд осуществляется еще на этапе загрузки кэш-памяти. Для каждого двойного слова кэш-память команд включает 6 дополнительных бит, которые содержат информацию о наличии зависимостей по данным и конфликтов ресурсов, что существенно упрощает выдачу команд в суперскалярном режиме.
В процессоре PA 7200 реализован эффективный алгоритм предварительной выборки команд, хорошо работающий и на линейных участках программ.
Как и в PA 7100 в процессоре реализован интерфейс с внешней кэш-памятью данных, работающей на тактовой частоте процессора с однотактным временем ожидания. Внешняя кэш-память данных построена по принципу прямого отображения. Кроме того, для повышения эффективности на кристалле процессора реализован небольшой вспомогательный кэш емкостью в 64 строки. Формирование, преобразование адреса и обращение к основной и вспомогательной кэш-памяти данных выполняется на двух ступенях конвейера. Максимальная задержка при обнаружении попадания равна одному такту.
Вспомогательный внутренний кэш содержит 64 32-байтовые строки. При обращении к кэш-памяти осуществляется проверка 65 тегов: 64-х тегов вспомогательного кэша и одного тега внешнего кэша данных. При обнаружении совпадения данные направляются в требуемое функциональное устройство.
При отсутствии необходимой строки в кэш-памяти производится ее загрузка из основной памяти. При этом строка поступает во вспомогательный кэш, что в ряде случаев позволяет сократить количество перезагрузок внешней кэш-памяти, организованной по принципу прямого отображения. Архитектурой нового процессора для команд загрузки/записи предусмотрено кодирование специального признака локального размещения данных ("spatial locality only"). При выполнении команд загрузки, помеченных этим признаком, происходит обычное заполнение строки вспомогательного кэша. Однако последующая запись строки осуществляется непосредственно в основную память минуя внешний кэш данных, что значительно повышает эффективность работы с большими массивами данных, для которых размера строки кэш-памяти с прямым отображением оказывается недостаточно.
Расширенный набор команд процессора позволяет реализовать средства автоиндексации для повышения эффективности работы с массивами, а также

осуществлять предварительную выборку команд, которые помещаются во вспомогательный внутренний кэш. Этот вспомогательный кэш обеспечивает динамическое расширение степени ассоциативности основной кэш-памяти, построенной на принципе прямого отображения, и является более простым альтернативным решением по сравнению с множественно-ассоциативной организацией.

Процессор PA 7200 включает интерфейс новой 64-битовой мультиплексной системной шины Runway, реализующей расщепление транзакций и поддержку протокола когерентности памяти. Этот интерфейс включает буфера транзакций, схемы арбитража и схемы управления соотношениями внешних и внутренних тактовых частот.

PA-8000

Процессор PA-8000 был анонсирован в марте 1995 года на конференции COMPCON 95. Было объявлено, что показатели его производительности будут достигать 8.6 единиц SPECint95 и 15 единиц SPECfp95 для операций целочисленной и вещественной арифметики соответственно. В настоящее время этот очень высокий уровень производительности подтвержден испытаниями рабочих станций и серверов, построенных на базе этого процессора.

Процессор PA-8000 вобрал в себя все известные методы ускорения выполнения команд. В его основе лежит концепция "интеллектуального выполнения", которая базируется на принципе внеочередного выполнения команд. Это свойство позволяет PA-8000 достигать пиковой суперскалярной производительности благодаря широкому использованию механизмов автоматического разрешения конфликтов по данным и управлению аппаратными средствами. Эти средства хорошо дополняют другие архитектурные компоненты, заложенные в структуру кристалла: большое число исполнительных функциональных устройств, средства прогнозирования направления переходов и выполнения команд по предположению, оптимизированная организация кэш-памяти и высокопроизводительный шинный интерфейс.

Высокая производительность PA-8000 во многом определяется наличием большого набора функциональных устройств, который включает в себя 10 исполнительных устройств: два арифметико-логических устройства (АЛУ) для выполнения целочисленных операций, два устройства для выполнения операций сдвига/слияния данных, два устройства для выполнения умножения/сложения чисел с плавающей точкой, два устройства деления/вычисления квадратного корня и два устройства выполнения операций загрузки/записи.

Средства внеочередного выполнения команд процессора PA-8000 обеспечивают аппаратное планирование загрузки конвейеров и лучшее использование функциональных устройств. В каждом такте на выполнение могут выдаваться до четырех команд, которые поступают в 56-строчный буфер переупорядочивания. Этот буфер позволяет поддерживать постоянную занятость функциональных устройств и обеспечивает эффективную минимизацию конфликтов по ресурсам. Кристалл может анализировать все 56 командных строк одновременно и выдавать в каждом такте по 4 готовых для выполнения команды в функциональные устройства. Это позволяет процессору автоматически выявлять параллелизм уровня выполнения команд.

Суперскалярный процессор PA-8000 обеспечивает полный набор средств выполнения 64-битовых операций, включая адресную арифметику, а также арифметику с фиксированной и плавающей точкой. При этом кристалл полностью сохраняет совместимость с 32-битовыми приложениями. Это первый процессор, в котором реализована 64-битовая архитектура PA-RISC. Он сохраняет полную совместимость с предыдущими и будущими реализациями PA-RISC.

Кристалл изготовлен по 0.5-микронной КМОП технологии с напряжением питания 3.3 В и можно рассчитывать на дальнейшее уменьшение размеров элементов в будущем.

8.4. 8.4. Особенности архитектуры MIPS компании MIPS Technology

Архитектура MIPS была одной из первых RISC-архитектур, получившей признание со стороны промышленности. Она была анонсирована в 1986 году. Первоначально это была полностью 32-битовая архитектура, которая включала 32 регистра общего назначения, 16 регистров плавающей точки и специальную пару регистров для хранения результатов выполнения операций целочисленного умножения и деления. Размер команд составлял 32 бит, в ней поддерживался всего один метод адресации, и пользовательское адресное пространство также определялось 32 битами. Выполнение арифметических операций регламентировалось стандартом IEEE 754. В компьютерной промышленности широкую популярность приобрели 32-битовые процессоры R2000 и R3000, которые в течение достаточно длительного времени служили основой для построения рабочих станций и серверов компаний Silicon Graphics, Digital, Siemens Nixdorf и др. Процессоры R3000/R3010 работали на тактовой частоте 33 или 40 МГц и обеспечивали производительность на уровне 20 SPECint92 и 23 SPECfp92.
Затем на смену микропроцессорам семейства R3000 пришли новые 64-битовые микропроцессоры R4000 и R4400. (MIPS Technology была первой компанией выпустившей процессоры с 64-битовой архитектурой). Набор команд этих процессоров (спецификация MIPS II) был расширен командами загрузки и записи 64-разрядных чисел с плавающей точкой, командами вычисления квадратного корня с одинарной и двойной точностью, командами условных прерываний, а также атомарными операциями, необходимыми для поддержки мультипроцессорных конфигураций. В процессорах R4000 и R4400 реализованы 64-битовые шины данных и 64-битовые регистры. В этих процессорах применяется метод удвоения внутренней тактовой частоты.
Процессоры R2000 и R3000 имели стандартные пятиступенчатые конвейеры команд. В процессорах R4000 и R4400 применяются более длинные конвейеры (иногда их называют суперконвейерами). Количество ступеней в процессорах R4000 и R4400 увеличилось до восьми, что объясняется прежде всего увеличением тактовой частоты и необходимостью распределения логики для обеспечения заданной пропускной способности конвейера. Процессор R4000 может работать с тактовой частотой 50/100 МГц и обеспечивает уровень производительности в 58 SPECint92 и 61 SPECfp92. Процессор R4400 может работать на частоте 50/100 МГц, или 75/150 МГц, показывая уровень производительности 94 SPECint92 и 105 SPECfp92.
Внутренняя кэш-память процессора R4000 имеет емкость 16 Кбайт. Она разделена на 8-Кб кэш команд и 8-Кб кэш данных. С точки зрения реализации кэш-памяти процессор R4400 имеет более развитые возможности. Он выпускается в трех модификациях: PC (Primary Cashe) - имеет внутренние кэши команд и данных емкостью по 16 Кбайт. Процессор в такой конфигурации предназначен главным образом для дешевых моделей рабочих станций. SC (Secondary Cashe) содержит логику управления кэш-памятью второго уровня. MC (Multiprocessor Cashe) - использует специальные алгоритмы обеспечения когерентности и согласованного состояния памяти для многопроцессорных конфигураций.
В середине 1994 года компания MIPS анонсировала процессор R8000, который прежде всего был ориентирован на научные прикладные задачи с интенсивным использованием операций с плавающей точкой. Этот процессор построен на двух кристаллах (выпускается в виде многокристальной сборки) и представляет собой первую суперскалярную реализацию архитектуры MIPS. Теоретическая пиковая производительность процессора для тактовой частоты 75 МГц составляет 300 MFLOPs (до четырех команд и шести операций с плавающей точкой в каждом такте). Реализация большой кэш-памяти данных емкостью 16 Мбайт, высокой пропускной способности доступа к данным (до 1.2 Гбайт/с) в сочетании с высокой скоростью

выполнения операций позволяет R8000 достигать 75% теоретической производительности даже при решении больших задач типа LINPACK с размерами матриц 1000x1000 элементов. Аппаратные средства поддержки когерентного состояния кэш-памяти вместе со средствами распараллеливания компиляторов обеспечивают возможность построения высокопроизводительных симметричных многопроцессорных систем. Например, процессоры R8000 используются в системе Power Challenge компании Silicon Graphics, которая вполне может сравниться по производительности с известными суперкомпьютерами Cray Y-MP, имеет на порядок меньшую стоимость и предъявляет значительно меньшие требования к подсистемам питания и охлаждения. В однопроцессорном исполнении эта система обеспечивает производительность на уровне 310 SPECfp92 и 265 MFLOPs на пакете LINPACK (1000x1000).
В 1994 году MIPS Technology объявила также о создании своего нового суперскалярного процессора R10000, начало массовых поставок которого ожидалось в конце 1995 года. По заявлениям представителей MIPS Technology R10000 обеспечивает пиковую производительность в 800 MIPS при работе с внутренней тактовой частотой 200 МГц за счет обеспечения выдачи для выполнения четырех команд в каждом такте синхронизации. При этом он обеспечивает обмен данными с кэш-памятью второго уровня со скоростью 3.2 Гбайт/с.
Чтобы обеспечить столь высокий уровень производительности в процессоре R10000 реализованы многие последние достижения в области технологии и архитектуры процессоров. На рисунке 8.13 показана блок-схема этого микропроцессора.

Иерархия памяти

При разработке процессора R10000 большое внимание было уделено эффективной реализации иерархии памяти. В нем обеспечиваются раннее обнаружение промахов кэш-памяти и параллельная перезагрузка строк с выполнением другой полезной работой. Реализованные на кристалле кэши поддерживают одновременную выборку команд, выполнение команд загрузки и записи данных в память, а также операций перезагрузки строк кэш-памяти. Заполнение строк кэш-памяти выполняется по принципу "запрошенное слово первым", что позволяет существенно сократить простои процессора из-за ожидания требуемой информации. Все кэши имеют двухканальную множественно-ассоциативную организацию с алгоритмом замещения LRU.

Кэш-память данных первого уровня

Кэш-память данных первого уровня процессора R10000 имеет емкость 32 Кбайт и организована в виде двух одинаковых банков емкостью по 16 Кбайт, что обеспечивает двухкратное расслоение при выполнении обращений к этой кэш-памяти. Каждый банк представляет собой двухканальную множественно-ассоциативную кэш-память с размером строки (блока) в 32 байта. Кэш данных индексируется с помощью виртуального адреса и хранит теги физических адресов памяти. Такой метод индексации позволяет выбрать подмножество кэш-памяти в том же такте, в котором формируется виртуальный адрес. Однако для того, чтобы поддерживать когерентность с кэш-памятью второго уровня, в кэше первого уровня хранятся теги физических адресов памяти.
Массивы данных и тегов в каждом банке являются независимыми. Эти четыре массива работают под общим управлением очереди формирования адресов памяти и схем внешнего интерфейса кристалла. В очереди адресов могут одновременно находиться до 16 команд загрузки и записи, которые обрабатываются в четырех отдельных конвейерах. Команды из этой очереди динамически выдаются для выполнения в специальный конвейер, который обеспечивает вычисление исполнительного виртуального адреса и преобразование этого адреса в физический. Три других параллельно работающих конвейера могут одновременно выполнять проверку тегов, осуществлять пересылку данных для команд загрузки и завершать выполнение команд записи в память. Хотя команды выполняются в строгом порядке их расположения в памяти, вычисление адресов и пересылка данных для команд загрузки могут

происходить неупорядоченно. Схемы внешнего интерфейса кристалла могут выполнять заполнение или обратное копирование строк кэш-памяти, либо операции просмотра тегов. Такая параллельная работа большинства устройств процессора позволяет R10000 эффективно выполнять реальные многопроцессорные приложения.

Работа конвейеров кэш-памяти данных тесно координирована. Например, команды загрузки могут выполнять проверку тегов и чтение данных в том же такте, что и преобразование адреса. Команды записи сразу же начинают проверку тегов, чтобы в случае необходимости как можно раньше инициировать заполнение требуемой строки из кэш-памяти второго уровня, но непосредственная запись данных в кэш задерживается до тех пор, пока сама команда записи не станет самой старой командой в общей очереди выполняемых команд и ей будет позволено зафиксировать свой результат ("выпустится"). Промах при обращении к кэш-памяти данных первого уровня инициирует процесс заполнения строки из кэш-памяти второго уровня. При выполнении команд загрузки одновременно с заполнением строки кэш-памяти данные могут поступать по цепям обхода в регистровый файл.

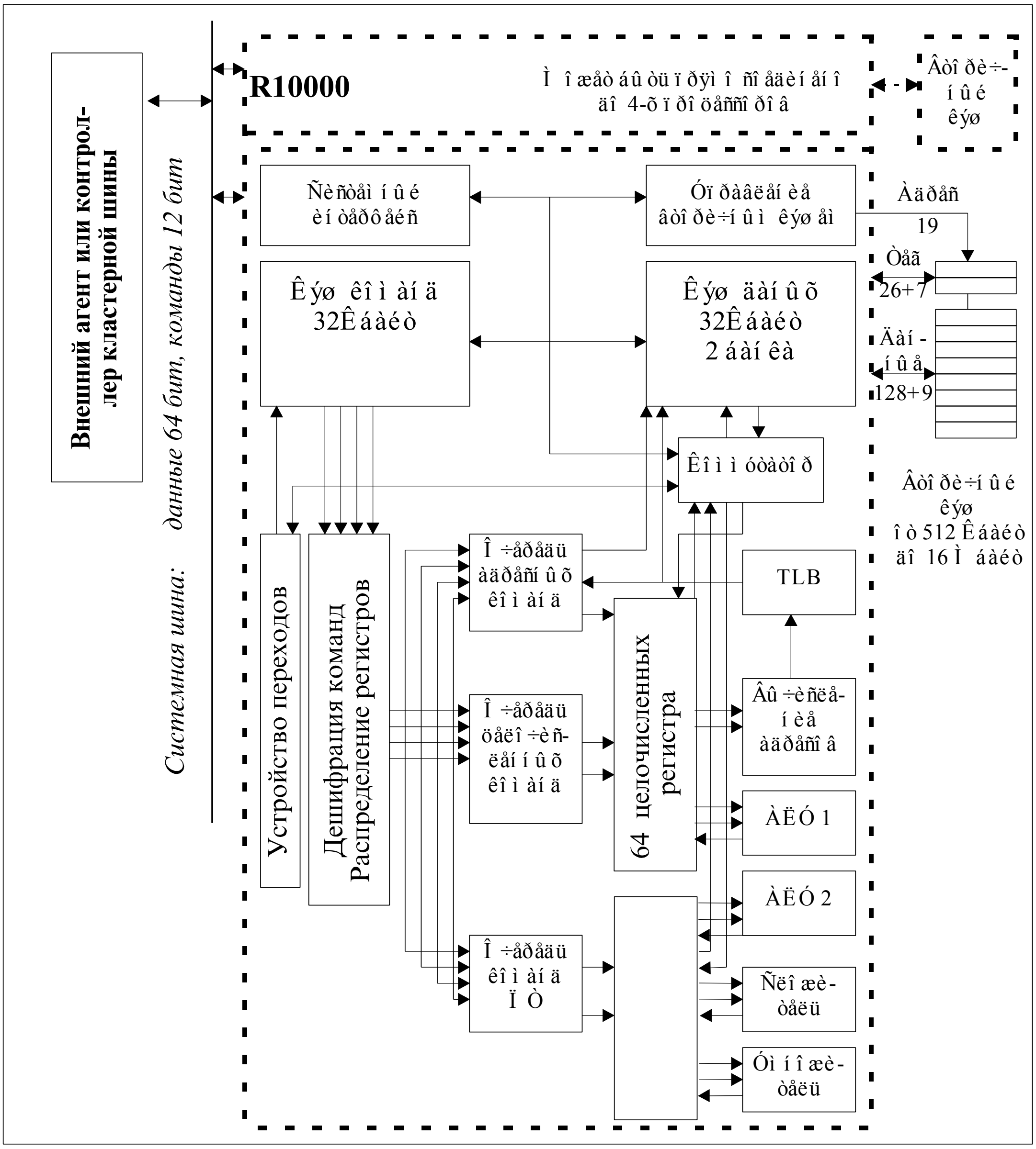


Рис. 8.13. Блок-схема микропроцессора R10000

При обнаружении промаха при обращении к кэш-памяти данных ее работа не блокируется, т.е. она может продолжать обслуживание следующих запросов. Это особенно полезно для уменьшения такого важного показателя качества реализованной архитектуры как среднее число тактов на команду (CPI - clock cycles per instruction). На рисунке 8.14 представлены результаты моделирования работы R10000 на нескольких программах тестового пакета SPEC. Для каждого теста даны два результата: с блокировкой кэш-памяти данных при обнаружении промаха (вверху) и действительное значение CPI R10000 (внизу). Выделенная более темным цветом правая область соответствует времени, потерянному из-за промахов кэш-памяти. Верхний результат отражает полную задержку в случае, если бы все операции по перезагрузке кэш-памяти выполнялись строго последовательно. Таким образом, стрелка представляет потери времени, которые возникают в блокируемом кэше. Эффект применения неблокируемой кэш-памяти сильно зависит характеристик самих программ. Для небольших тестов, рабочие наборы которых полностью помещаются в кэш-памяти первого уровня, этот эффект не велик. Однако для более реальных программ, подобных тесту tomcatv или тяжелому для кэш-памяти тесту compress, выигрыш оказывается существенным.

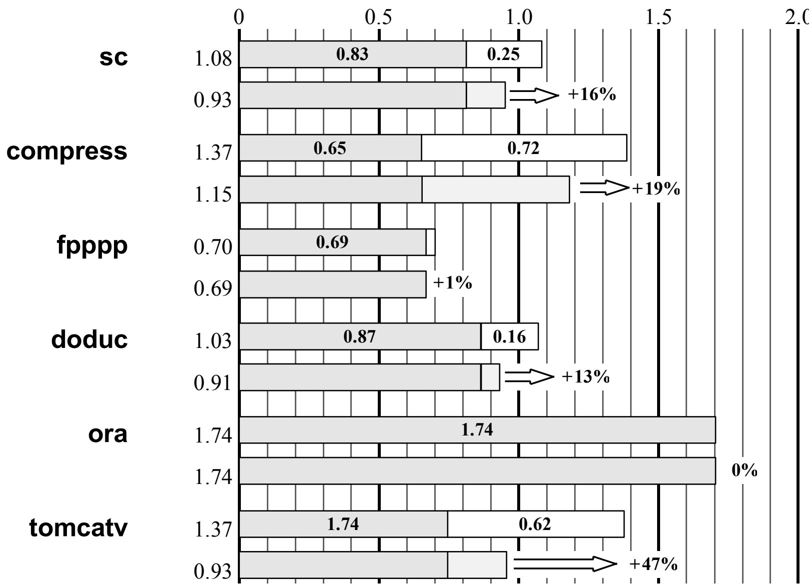


Рис. 8.14. Моделирование работы R10000 на нескольких компонентах пакета SPEC

Кэш-память второго уровня

Интерфейс кэш-памяти второго уровня процессора R10000 поддерживает 128-битовую магистраль данных, которая может работать с тактовой частотой до 200 МГц, обеспечивая скорость обмена до 3.2 Гбайт/с (для снижения требований к быстродействию микросхем памяти предусмотрена также возможность деления частоты с коэффициентами 1.5, 2, 2.5 и 3). Все стандартные синхронные сигналы управления статической памятью вырабатываются внутри процессора. Не требуется никаких внешних интерфейсных схем. Минимальный объем кэш-памяти второго уровня составляет 512 Кбайт, максимальный размер - 16 Мбайт. Размер строки этой кэш-памяти программируется и может составлять 64 или 128 байт.

Одним из методов улучшения временных показателей работы кэш-памяти является построение псевдо-множествнно-ассоциативной кэш-памяти. В такой кэш-памяти частота промахов находится на уровне частоты промахов множественно-ассоциативной памяти, а время выборки при попадании соответствует кэш-памяти с прямым отображением. Кэш-память R10000 организована именно таким способом, причем для ее реализации используются стандартные синхронные микросхемы памяти (SRAM). В одном наборе микросхем памяти находятся оба канала кэша. Информация о частоте использования этих каналов хранится в схемах управления кэшем на процессорном кристалле. Поэтому после обнаружения промаха в первичном кэше из наиболее часто используемого канала вторичного кэша считываются две четырехсловные строки. Их теги считываются вместе с первой четырехсловной строкой, а теги альтернативного канала читаются одновременно со второй четырехсловной строкой (это осуществляется простым инвертированием старшего разряда адреса).

При этом возможны три случая. Если происходит попадание по первому каналу, то данные доступны немедленно. Если происходит попадание по альтернативному каналу, происходит повторное чтение вторичного кэша. Если отсутствует попадание по обоим каналам, вторичный кэш должен перезаполняться из основной памяти.

Для обеспечения целостности данных в кэш-памяти большой емкости обычной практикой является использование кодов исправляющих одиночные ошибки (ECC-кодов). В R10000 с каждой четырехсловной строкой хранится 9-битовый ECC-код и бит четности. Дополнительный бит четности позволяет сократить задержку, поскольку проверка на четность может быть выполнена очень быстро, чтобы предотвратить использование некорректных данных. При этом, если обнаруживается корректируемая ошибка, то чтение повторяется через специальный двухтактный конвейер коррекции ошибок.

Кэш-память команд

Объем внутренней двухканальной множественно-ассоциативной кэш-памяти команд составляет 32 Кбайт. В процессе ее загрузки команды частично декодируются. При этом к каждой команде добавляются 4 дополнительных бит, которые указывают исполнительное устройство, в котором она будет выполняться. Таким образом, в кэш-памяти команды хранятся в 36-битовом формате. Размер строки кэш-памяти команд составляет 64 байта.

Обработка команд перехода

При реализации конвейерной обработки возникают ситуации, которые препятствуют выполнению очередной команды из потока команд в предназначенном для нее такте. Такие ситуации называются конфликтами. Конфликты снижают реальную производительность конвейера, которая могла бы быть достигнута в идеальном случае. Одним из типов конфликтов, с которыми приходится иметь дело разработчикам высокопроизводительных процессоров, являются конфликты по управлению, которые возникают при конвейеризации команд перехода и других команд, изменяющих значение счетчика команд.

Конфликты по управлению могут вызывать даже большие потери производительности суперскалярного процессора, чем конфликты по данным. По статистике среди команд управления, меняющих значение счетчика команд, преобладают команды условного перехода. Таким образом, снижение потерь от условных переходов становится критически важным вопросом. Имеется несколько методов сокращения приостановок конвейера, возникающих из-за задержек выполнения условных переходов. В процессоре R10000 используются два наиболее мощных метода динамической оптимизации выполнения условных переходов: аппаратное прогнозирование направления условных переходов и "выполнение по предположению" (speculation).

Устройство переходов процессора R10000 может декодировать и выполнять только по одной команде перехода в каждом такте. Поскольку за каждой командой перехода следует слот задержки, максимально могут быть одновременно выбраны две команды перехода, но только одна более ранняя команда перехода может декодироваться в данный момент времени. Во время декодирования команд к каждой команде добавляется бит признака перехода. Эти биты используются для пометки команд перехода в конвейере выборки команд.

Направление условного перехода прогнозируется с помощью специальной памяти (branch history table) емкостью 512 строк, которая хранит историю выполнения переходов в прошлом. Обращение к этой таблице осуществляется с помощью адреса команды во время ее выборки. Двухбитовый код прогноза в этой памяти обновляется каждый раз, когда принято окончательное решение о направлении перехода. Моделирование показало, что точность двухбитовой схемы прогнозирования для тестового пакета программ SPEC составляет 87%.

Все команды, выбранные вслед за командой условного перехода, считаются выполняемыми по предположению (условно). Это означает, что в момент их выборки заранее не известно, будет ли завершено их выполнение. Процессор допускает предварительную обработку и прогнозирование направления четырех команд условного перехода, которые могут разрешаться в произвольном порядке. При этом для каждой выполняемой по предположению команды условного перехода в специальный стек переходов записывается информация, необходимая для восстановления состояния процессора в случае, если направление перехода было предсказано неверно. Стек переходов имеет глубину в 4 элемента и позволяет в случае необходимости быстро и эффективно (за один такт) восстановить конвейер.

Структура очередей команд

Процессор R10000 содержит три очереди (буфера) команд (очередь целочисленных команд, очередь команд плавающей точки и адресную очередь). Эти три очереди осуществляют динамическую выдачу команд в соответствующие исполнительные устройства. С каждой командой в очереди хранится тег команды, который перемещается вместе с командой по ступеням конвейера. Каждая очередь осуществляет динамическое планирование потока команд и может определить моменты времени, когда становятся доступными операнды, необходимые для выполнения каждой команды. Кроме того, очередь определяет порядок выполнения команд на основе анализа состояния соответствующих исполнительных устройств. Как только ресурс оказывается свободным очередь выдает команду в соответствующее исполнительное устройство.

Очередь целочисленных команд

Очередь целочисленных команд содержит 16 строк и выдает команды в два арифметико-логических устройства. Целочисленные команды поступают в свободные строки этой очереди, причем в каждом такте в нее могут записываться до 4 команд. Целочисленные команды остаются в очереди до тех пор, пока они не будут выданы в одно из АЛУ.

Очередь команд плавающей точки

Очередь команд плавающей точки также содержит 16 строк и выдает команды в исполнительные устройства сложения и умножения с плавающей точкой. Команды плавающей точки поступают в свободные строки очереди, причем в каждом такте в нее могут записываться до 4 команд. Команды остаются в очереди до тех пор, пока они не будут выданы в одно из исполнительных устройств. Очередь команд плавающей точки содержит также логику управления команд типа "умножить-сложить". Эта команда сначала направляется в устройство умножения, а затем прямо в устройство сложения.

Адресная очередь

Очередь адресных команд выдает команды в устройство загрузки/записи и содержит 16 строк. Очередь организована в виде циклического буфера FIFO (first-in first-out). Команды могут выдаваться в произвольном порядке, но должны записываться в очередь и изыматься из нее строго последовательно. В каждом такте в очередь могут поступать до 4 команд. Буфер FIFO поддерживает первоначальную последовательность команд, что упрощает обнаружение зависимостей по адресам. Выполнение выданной команды может не закончиться при обнаружении зависимости по адресам, кэш-промаха или конфликта по ресурсам. В этих случаях адресная очередь должна заново повторять выдачу команды до тех пор, пока ее выполнение не завершится.

Переименование регистров

Одним из аппаратных методов минимизации конфликтов по данным является метод переименования регистров (register renaming). Он получил свое название от широко применяющегося в компиляторах метода переименования - метода размещения данных, способствующего сокращению числа зависимостей и тем самым увеличению производительности при отображении необходимых исходной программе объектов (например, переменных) на аппаратные ресурсы (например, ячейки памяти и регистры).

При аппаратной реализации метода переименования регистров выделяются логические регистры, обращение к которым выполняется с помощью соответствующих полей команды, и физические регистры, которые размещаются в аппаратном регистровом файле процессора. Номера логических регистров динамически отображаются на номера физических регистров посредством таблиц отображения, которые обновляются после декодирования каждой команды. Каждый новый результат записывается в новый физический регистр. Однако предыдущее значение каждого логического регистра сохраняется и может быть восстановлено в случае, если выполнение команды должно быть прервано из-за возникновения исключительной ситуации или неправильного предсказания направления условного перехода.

В процессе выполнения программы генерируется множество временных регистровых результатов. Эти временные значения записываются в регистровые файлы вместе с постоянными значениями. Временное значение становится новым постоянным значением, когда завершается выполнение команды (фиксируется ее результат). В свою очередь, завершение выполнения команды происходит, когда все предыдущие команды успешно завершились в заданном программой порядке. Программист (или компилятор) имеет дело только с логическими регистрами. Реализация физических регистров от него скрыта.

Таким образом, аппаратный метод переименования регистров, используемый в процессоре R10000, имеет три основных достоинства. Во-первых, результаты "выполняемых по предположению" команд могут прямо записываться в регистровый файл. Во-вторых, этот метод устраняет все конфликты типа "запись после чтения" и "запись после записи", которые часто возникают при неупорядоченном выполнении команд. И, наконец, метод переименования регистров упрощает контроль зависимостей по данным. Поскольку процессор обеспечивает выдачу для выполнения до четырех команд в каждом такте, в процессе переименования регистров их логические номера сравниваются для определения зависимостей между четырьмя командами, декодированными в одном и том же такте.

Реализованная в микропроцессоре R10000 схема отображения команд состоит из двух таблиц отображения, списка активных команд и двух списков свободных регистров (для целочисленных команд и команд плавающей точки имеются отдельные таблицы отображения и списки свободных регистров). Чтобы поддерживать последовательный порядок завершения выполнения команд, существует только один список активных команд, который содержит как целочисленные команды, так и команды плавающей точки.

Микропроцессор R10000 содержит по 64 физических регистра (целочисленных и плавающей точки). В любой момент времени значение физического регистра содержится в одном из указанных выше списков. На рисунке 8.15 показана упрощенная блок-схема отображения целочисленных команд.

Команды выбираются из кэша команд и помещаются в таблицу отображения. В любой момент времени каждый из 64 номеров физических регистров находится в одном из трех указанных на рисунке блоков.

Список активных команд длиною 32 элемента может хранить упорядоченную в соответствии с программой последовательность команд, которые могут находиться в обработке в любой данный момент времени. Команды из очереди целочисленных команд могут выполняться неупорядочено и записывать результаты в физические регистры, но порядок их окончательного завершения определяется списком активных команд.

Каждая команда может уникально идентифицироваться своим положением в списке активных команд. Поэтому каждую команду в очереди и в соответствующем исполнительном устройстве сопровождает 5-битовая метка, называемая тегом команды. Этот тег и определяет положение команды в списке активных команд. Когда в исполнительном устройстве заканчивается выполнение команды, тег позволяет очень просто ее отыскать в списке активных команд и пометить как выполненную. Когда результат операции из исполнительного устройства записывается в физический регистр, номер этого физического регистра становится больше не нужным и может быть затем возвращен в список свободных регистров, а соответствующая команда перестает быть активной.

Когда в процессе переименования из списка свободных регистров выбирается очередной номер физического регистра, он передается в таблицу отображения, которая обновляется. При этом старый номер регистра, соответствующий определенному в команде логическому регистру результата, помещается из таблицы отображения в список активных команд. Этот номер остается в списке активных команд до тех пор, пока соответствующая команда не "выпустится" (graduate), т.е. завершится в заданном программой порядке. Команда может "выпуститься" только после того, как успешно завершится выполнение всех предыдущих команд.

Микропроцессор R10000 содержит 64 физических и 32 логических целочисленных регистра. Список активных команд может содержать максимально 32 элемента. Список свободных регистров также может максимально содержать 32 значения. Если список активных команд полон, то могут быть 32 "зафиксированных" и 32 временных значения. Отсюда потребность в 64 регистрах.

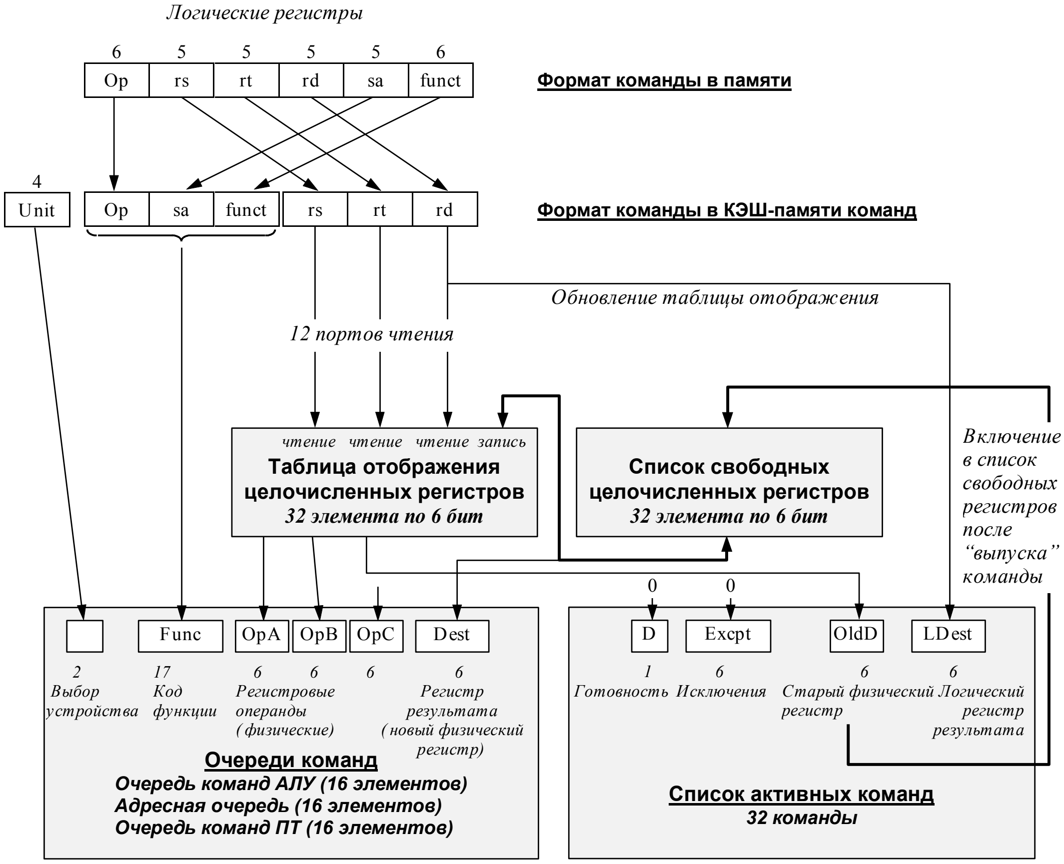


Рис. 8.15. Упрощенная блок-схема отображения целочисленных команд

Исполнительные устройства

В процессоре R10000 имеются пять полностью независимых исполнительных устройств: два целочисленных АЛУ, два основных устройства плавающей точки с двумя вторичными устройствами плавающей точки, которые работают с длинными операциями деления и вычисления квадратного корня, а также устройство загрузки/записи.

Целочисленные АЛУ

В микропроцессоре R10000 имеются два целочисленных АЛУ: АЛУ1 и АЛУ2. Время выполнения всех целочисленных операций АЛУ (за исключением операций умножения и деления) и частота повторений составляют один такт.

Оба АЛУ выполняют стандартные операции сложения, вычитания и логические операции. Эти операции завершаются за один такт. АЛУ1 обрабатывает все команды перехода, а также операции сдвига, а АЛУ2 - все операции умножения и деления с использованием итерационных алгоритмов. Целочисленные операции умножения и деления помещают свои результаты в регистры EntryHi и EntryLo.

Во время выполнения операций умножения в АЛУ2 могут выполняться другие однотактные команды, но сам умножитель оказывается занятым. Однако когда умножитель заканчивает свою работу, АЛУ2 оказывается занятым на два такта, чтобы обеспечить запись результата в два регистра. Во время выполнения операций деления, которые имеют очень большую задержку, АЛУ2 занято на все время выполнения операции.

Целочисленные операции умножения вырабатывают произведение с двойной точностью. Для операций с одинарной точностью происходит распространение знака результата до 64 бит прежде, чем он будет помещен в регистры EntryHi и EntryLo. Время выполнения операций с двойной точностью примерно в два раза превосходит время выполнения операций с одинарной точностью.

Устройства плавающей точки

В микропроцессоре R10000 реализованы два основных устройства плавающей точки. Устройство сложения обрабатывает операции сложения, а устройство умножения - операции умножения. Кроме того, существуют два вторичных устройства плавающей точки, которые обрабатывают длинные операции деления и вычисления квадратного корня.

Время выполнения команд сложения, вычитания и преобразования типов равно двум тактам, а скорость их поступления в устройство составляет 1 команда/такт. Эти команды обрабатываются в устройстве сложения. Команды преобразования целочисленных значений в значения с плавающей точкой с однократной точностью имеют задержку в 4 такта, поскольку они должны пройти через устройство сложения дважды.

В устройстве умножения обрабатываются все операции умножения с плавающей точкой. Время их выполнения составляет два такта, а скорость поступления - 1 команда/такт. Устройства деления и вычисления квадратного корня выполняют операции с использованием итерационных алгоритмов. Эти устройства не конвейеризованы и не могут начать выполнение следующей операции до тех пор, пока не завершилось выполнение текущей команды. Таким образом, скорость повторения этих операций примерно равна задержке их выполнения. Порты умножителя являются общими и для устройств деления и вычисления квадратного корня. В начале и в конце операции теряется по одному такту (для выборки операндов и для записи результата).

Операция с плавающей точкой "умножить-сложить", которая в вычислительных программах возникает достаточно часто, выполняется с использованием двух

отдельных операций: операции умножения и операции сложения. Команда "умножить-сложить" (MADD) имеет задержку 4 такта и скорость повторения 1 команда/ такт. Эта составная команда увеличивает производительность за счет устранения выборки и декодирования дополнительной команды.
Устройства деления и вычисления квадратного корня используют раздельные цепи и могут работать одновременно. Однако очередь команд плавающей точки не может выдать для выполнения обе команды в одном и том же такте.

Устройство загрузки/записи и TLB
Устройство загрузки/записи содержит очередь адресов, устройство вычисления адреса, устройство преобразования виртуальных адресов в физические (TLB), стек адресов, буфер записи и кэш-память данных первого уровня. Устройство загрузки/записи выполняет команды загрузки, записи, предварительной выборки, а также команды работы с кэш-памятью.
Выполнение всех команд загрузки и записи начинается с трехтактной последовательности, во время которой осуществляется выдача команды, вычисление виртуального адреса и его преобразование в физический. Преобразование адреса осуществляется во время выполнения команды только однажды. Производится обращение к кэш-памяти данных, и пересылка требуемых данных завершается при наличии данных в кэш-памяти первого уровня.
В случае промаха, или в случае занятости разделяемого порта регистрового файла, обращение к кэшу данных и к тегу должно быть повторено после получения данных либо из кэш-памяти второго уровня, либо из основной памяти.
TLB содержит 64 строки и выполняет преобразование виртуального адреса в физический. Виртуальный адрес для преобразования поступает либо из устройства вычисления адреса, либо из счетчика команд.
Интерфейс кэш-памяти второго уровня
Внешняя кэш-память второго уровня управляется с помощью внутреннего контроллера, который имеет специальный порт для подсоединения кэш-памяти. Специальная магистраль данных шириной в 128 бит осуществляет пересылки данных на внутренней тактовой частоте процессора 200 МГц, обеспечивая максимальную скорость передачи данных кэш-памяти второго уровня 3.2 Гбайт/с. В процессоре имеется также 64-битовая шина данных системного интерфейса.
Кэш-память второго уровня имеет двухканальную множественно-ассоциативную организацию. Максимальный размер этой кэш-памяти - 16 Мбайт. Минимальный размер - 512 Кбайт. Пересылки осуществляются 128-битовыми порциями (4 32-битовых слова). Для пересылки больших блоков данных используются последовательные циклы шины:

- • Четырехсловные обращения (128 бит) используются для команд кэш-памяти (CASHE);
- • Восьмисловные обращения (256 бит) используются для перезагрузки первичного кэша данных;
- • Шестнадцатисловные обращения (512 бит) используются для перезагрузки первичного кэша команд;
- • Тридцатидвухсловные обращения (1024 бит) используются для перезагрузки кэш-памяти второго уровня.

Системный интерфейс
Системный интерфейс процессора R10000 работает в качестве шлюза между самим процессором, связанным с ним кэшем второго уровня и остальной системой. Системный интерфейс работает с тактовой частотой внешней синхронизации (SysClk). Возможно программирование работы системного интерфейса на тактовой частоте 200, 133, 100, 80,

67, 57 и 50 МГц. Все выходы и входы системного интерфейса синхронизируются нарастающим фронтом сигнала SysClk, позволяя ему работать на максимально возможной тактовой частоте.

В большинстве микропроцессорных систем в каждый момент времени может происходить только одна системная транзакция.

Процессор R10000 поддерживает протокол расщепления транзакций, позволяющий осуществлять выдачу очередных запросов процессором или внешним абонентом шины, не дожидаясь ответа на предыдущий запрос. Максимально в любой момент времени поддерживается до четырех одновременных транзакций на шине.

Поддержка многопроцессорной организации

Процессор R10000 допускает два способа организации многопроцессорной системы. Один из способов связан с созданием специального внешнего интерфейса (агента) для каждого процессора системы. Этот интерфейс обычно реализуется с помощью заказной интегральной схемы, которая организует шлюз к основной памяти и подсистеме ввода/вывода. При таком типе соединений процессоры не связаны друг с другом непосредственно, а взаимодействуют через этот специальный интерфейс. Хотя такая реализация общепринята, ее стоимость, а также общая сложность системы достаточно высоки.(поскольку по крайней мере один внешний агент должен сопровождать каждый процессор.

Второй способ предназначен для достижения максимальной производительности минимальными затратами. Он подразумевает использование от двух до четырех процессоров, объединенных шиной Claster Bus. В этом случае необходим только один внешний интерфейс для взаимодействия с другими ресурсами системы. Все процессоры связаны с одним и тем же внешним агентом. Реализация кластерной шины не только снижает сложность, но и количество заказных интегральных схем, а следовательно и стоимость системы, требуя только одного внешнего агента на каждые четыре процессора.

В дополнение к 64-битовой мультиплексированной шины адреса/данных имеется двухбитовая шина состояний, которая используется для выдачи ответов о состоянии процессорной когерентности. Кроме того, используется 5-битовая шина системных ответов внешним агентом для выдачи внешних ответов подтверждения. На рисунке 8.16 показана блок-схема конфигурации кластерной шины.

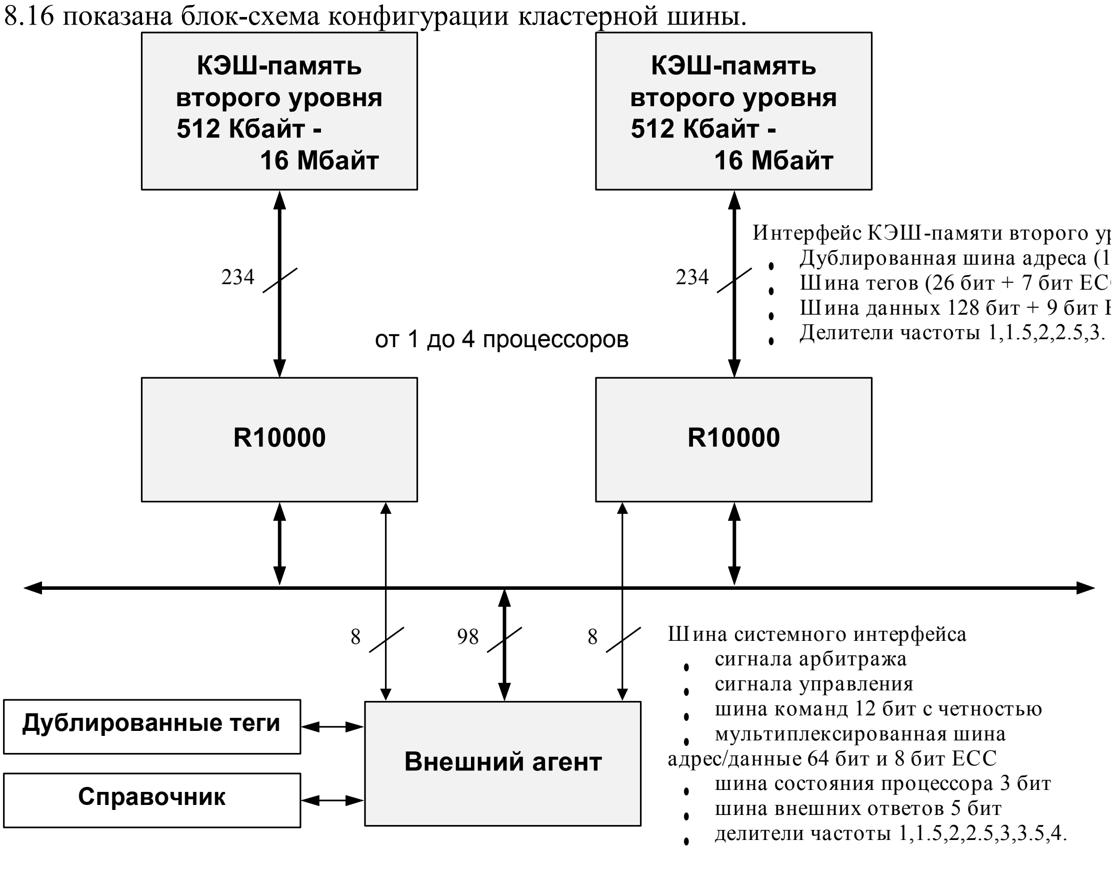


Рис. 8.16. Построение многопроцессорной системы на базе кластерной шины

8.5. 8.5. Особенности архитектуры Alpha компании DEC

В настоящее время семейство микропроцессоров с архитектурой Alpha представлено несколькими кристаллами, имеющими различные диапазоны производительности, работающие с разной тактовой частотой и рассеивающие разную мощность.

Первым на рынке появился 64-разрядный микропроцессор Alpha (DECchip 21064) . Он представляет собой RISC-процессор в однокристальном исполнении, в состав которого входят устройства целочисленной и плавающей арифметики, а также кэш-память емкостью 16 Кб. Кристалл проектировался с учетом реализации передовых методов увеличения производительности, включая конвейерную организацию всех функциональных устройств, одновременную выдачу нескольких команд для выполнения, а также средства организации симметричной многопроцессорной обработки.

В кристалле имеются два регистровых файла по 32 64-битовых регистра: один для целых чисел, второй - для чисел с плавающей точкой. Для обеспечения совместимости с архитектурами MIPS и VAX архитектура Alpha поддерживает арифметику с одинарной и двойной точностью как в соответствии со стандартом IEEE 754, так и в соответствии с внутренним для компании стандартом арифметики VAX.

Самая мощная модель процессора 21064 работает на частоте 200 МГц. В конце 1993 года появилась модернизированная версия кристалла - модель 21064А, имеющая на кристалле кэш-память удвоенного объема и работающая с тактовой частотой 275 МГц.

Затем были выпущены модели 21066 и 21068, оперирующие на частоте 166 и 66 МГц. Отличительной особенностью этой ветви процессоров Alpha является реализация на кристалле шины PCI. Это существенно упрощает и удешевляет как проектирование, так и производство компьютеров. Отличительная особенность модели 21068 - низкая потребляемая мощность (около 8 ватт). Основное предназначение этих двух новых моделей - персональные компьютеры и одноплатные ЭВМ.

На рисунке 8.17 представлена блок-схема микропроцессора 21066. Основными компонентами этого процессора являются: кэш-память команд, целочисленное устройство, устройство плавающей точки, устройство выполнения команд загрузки/записи, кэш-память данных, а также контроллер памяти и контроллер ввода/вывода.

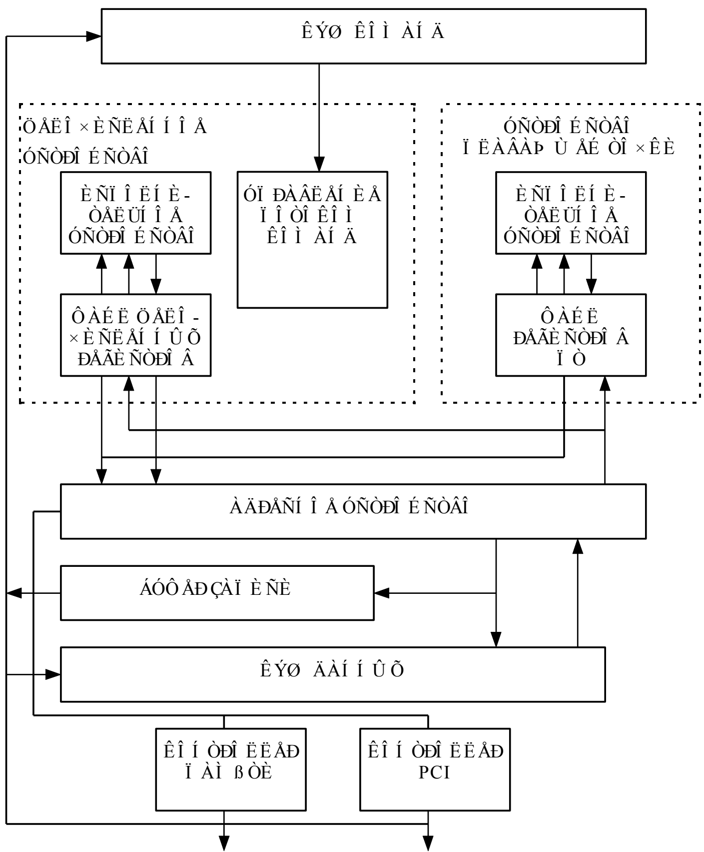


Рис. 8.17. Основные компоненты процессора Alpha 21066

Кэш-память команд представляет собой кэш прямого отображения емкостью 8 Кбайт. Команды, выбираемые из этой кэш-памяти, могут выдаваться попарно для выполнения в одно из исполнительных устройств. Кэш-память данных емкостью 8 Кбайт также реализует кэш с прямым отображением. При выполнении операций записи в память данные одновременно записываются в этот кэш и в буфер записи. Контроллер памяти или контроллер ввода/вывода шины PCI обрабатывают все обращения, которые проходят через расположенные на кристалле кэш-памяти первого уровня.

Контроллер памяти прежде всего проверяет содержимое внешней кэш-памяти второго уровня, которая построена на принципе прямого отображения и реализует алгоритм отложенного обратного копирования при выполнении операций записи. При обнаружении промаха контроллер обращается к основной памяти для перезагрузки соответствующих строк кэш-памяти. Контроллер ввода/вывода шины PCI обрабатывает весь трафик, связанный с вводом/выводом. Под управлением центрального процессора он выполняет операции программируемого ввода/вывода. Трафик прямого доступа к памяти шины PCI обрабатывается контроллером PCI совместно с контроллером памяти. При выполнении операций прямого доступа к памяти в режиме чтения и записи данные не размещаются в кэш-памяти второго уровня. Интерфейсы памяти и PCI были разработаны специально в расчете на однопроцессорные конфигурации и не поддерживают реализацию мультипроцессорной архитектуры.

На рисунке 8.18 показан пример системы, построенной на базе микропроцессора 21066. В представленной конфигурации контроллер памяти выполняет обращения как к статической памяти, с помощью которой реализована кэш-память второго уровня, так и к динамической памяти, на которой построена основная память. Для хранения тегов и данных в кэш-памяти второго уровня используются кристаллы статической памяти с одинаковым временем доступа по чтению и записи.

Конструкция поддерживает до четырех банков динамической памяти, каждый из которых может управляться независимо, что дает определенную гибкость при организации памяти и ее модернизации. Один из банков может заполняться микросхемами видеопамяти (VRAM) для реализации дешевой графики. Контроллер памяти прямо работает с видеопамятью и поддерживает несколько простых графических операций.

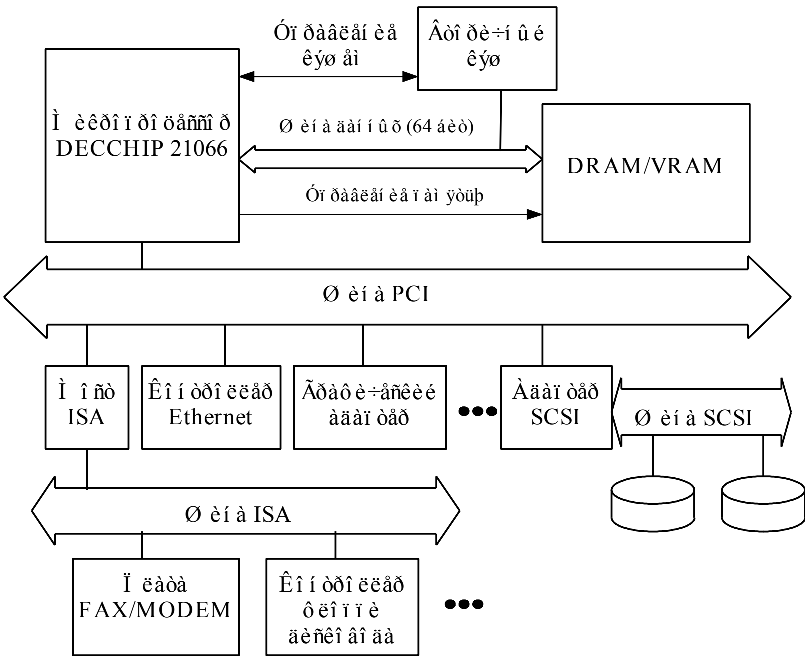


Рис. 8.18. Пример построения системы на базе микропроцессора Alpha 21066

Высокоскоростная шина PCI имеет ряд привлекательных свойств. Помимо возможности работы с прямым доступом к памяти и программируемым вводом/выводом она допускает специальные конфигурационные циклы, расширяемость до 64 бит, компоненты, работающие с питающими напряжениями 3.3 и 5 В, а также более быстрое тактирование. Базовая реализация шины PCI поддерживает мультиплексирование адреса и данных и работает на частоте 33 МГц, обеспечивая максимальную скорость передачи данных 132 Мбайт/с. Шина PCI непосредственно управляется микропроцессором. На рисунке 8.18 показаны некоторые высокоскоростные периферийные устройства: графические адаптеры, контроллеры SCSI и сетевые адаптеры, подключенные непосредственно к шине PCI. Мостовая микросхема интерфейса ISA позволяет подключить к системе низкоскоростные устройства типа модема, флоппи-дисковода и т.д.

В настоящее время выпущена модернизированная версия этого микропроцессора. Как и его предшественник, новый кристалл Alpha 21066A помимо интерфейса PCI содержит на кристалле интегрированный контроллер памяти и графический акселератор. Эти характеристики позволяют значительно снизить стоимость реализации систем, базирующихся на Alpha 21066A, и обеспечивают простой и дешевый доступ к внешней памяти и периферийным устройствам. Alpha 21066A имеет две модификации в соответствии с частотой: 100 МГц и 233 МГц. Модель с 233 МГц обеспечивает производительность 94 и 100 единиц, соответственно, по тестам SPECint92 и SPECfp92.

Новейший микропроцессор Alpha 21164 представляет собой вторую полностью новую реализацию архитектуры Alpha. Микропроцессор 21164, представленный в сентябре 1994 года, обеспечивает производительность 330 и 500 единиц, соответственно, по шкалам SPECint92 и SPECfp92 или около 1200 MIPS и выполняет до четырех инструкций за такт. На кристалле микропроцессора 21164 размещено около 9,3 миллиона транзисторов, большинство из которых образуют кэш. Кристалл построен на базе 0.5 микронной КМОП технологии компании DEC. Он собирается в 499-контактные корпуса PGA (при этом 205 контактов отводятся под разводку питания и земли) и рассеивает 50 Вт при питающем напряжении 3.3 В на частоте 300 МГц.

Переход в 1996 году на 0.35 микронную КМОП технологию привел к возможности дальнейшего увеличения тактовой частоты и производительности процессора. В настоящее время процессоры 21164 выпускаются с тактовой частотой 366 МГц (11.3 SPECint95, 15.4 SPECfp95) и 433 МГц (13.3 SPECint95, 18.3 SPECfp95). В конце 1996 года начнутся массовые поставки 21164 с тактовой частотой 500 МГц (15.4 SPECint95, 21.1 SPECfp95). Таким образом, компания DEC в настоящее время имеет самые мощные процессоры, пиковая производительность которых составляет 2 миллиарда операций в секунду.

Ключевыми моментами для реализации высокой производительности является суперскалярный режим работы процессора, обеспечивающий выдачу для выполнения до четырех команд в каждом такте, высокопроизводительная неблокируемая подсистема памяти с быстродействующей кэш-памятью первого уровня, большая, размещенная на кристалле, кэш-память второго уровня и уменьшенная задержка выполнения операций во всех функциональных устройствах.

На рисунке 8.19 представлена блок-схема процессора, который включает пять функциональных устройств: устройство управления потоком команд (IBOX), целочисленное устройство (EBOX), устройство плавающей точки (FBOX), устройство управления памятью (MBOX) и устройство управления кэш-памятью и интерфейсом шины (CBOX). На рисунке также показаны три расположенных на кристалле кэш-памяти. Кэш-память команд и кэш-память данных представляют собой первичные кэши, реализующие прямое отображение. Множественно-ассоциативная кэш-память второго уровня предназначена для хранения команд и данных. Длина конвейеров процессора 21164 варьируется от 7 ступеней для выполнения целочисленных команд и 9 ступеней для реализации команд с плавающей точкой до 12 ступеней при выполнении команд обращения к памяти в пределах кристалла и переменного числа ступеней при выполнении команд обращения к памяти за пределами кристалла.

Устройство управления потоком команд осуществляет выборку и декодирование команд из кэша команд и направляет их для выполнения в соответствующие исполнительные устройства после разрешения всех конфликтов по регистрам и функциональным устройствам. Оно управляет выполнением программы и всеми аспектами обработки исключительных ситуаций, ловушек и прерываний. Кроме того, оно обеспечивает управление всеми исполнительными устройствами, контролируя все цепи обхода данных и записи в регистровый файл. Устройство управления содержит 8 Кбайт кэш команд, схемы предварительной выборки команд и связанный с ними буфер перезагрузки, схемы прогнозирования направления условных переходов и буфер преобразования адресов команд (ITB).

Целочисленное исполнительное устройство выполняет целочисленные команды, вычисляет виртуальные адреса для всех команд загрузки и записи, выполняет целочисленные команды условного перехода и все другие команды управления. Оно включает в себя регистровый файл и несколько функциональных устройств, расположенных на четырех ступенях двух параллельных конвейеров. Первый конвейер содержит сумматор, устройство логических операций, сдвигатель и умножитель. Второй конвейер содержит сумматор, устройство логических операций и устройство выполнения команд управления.

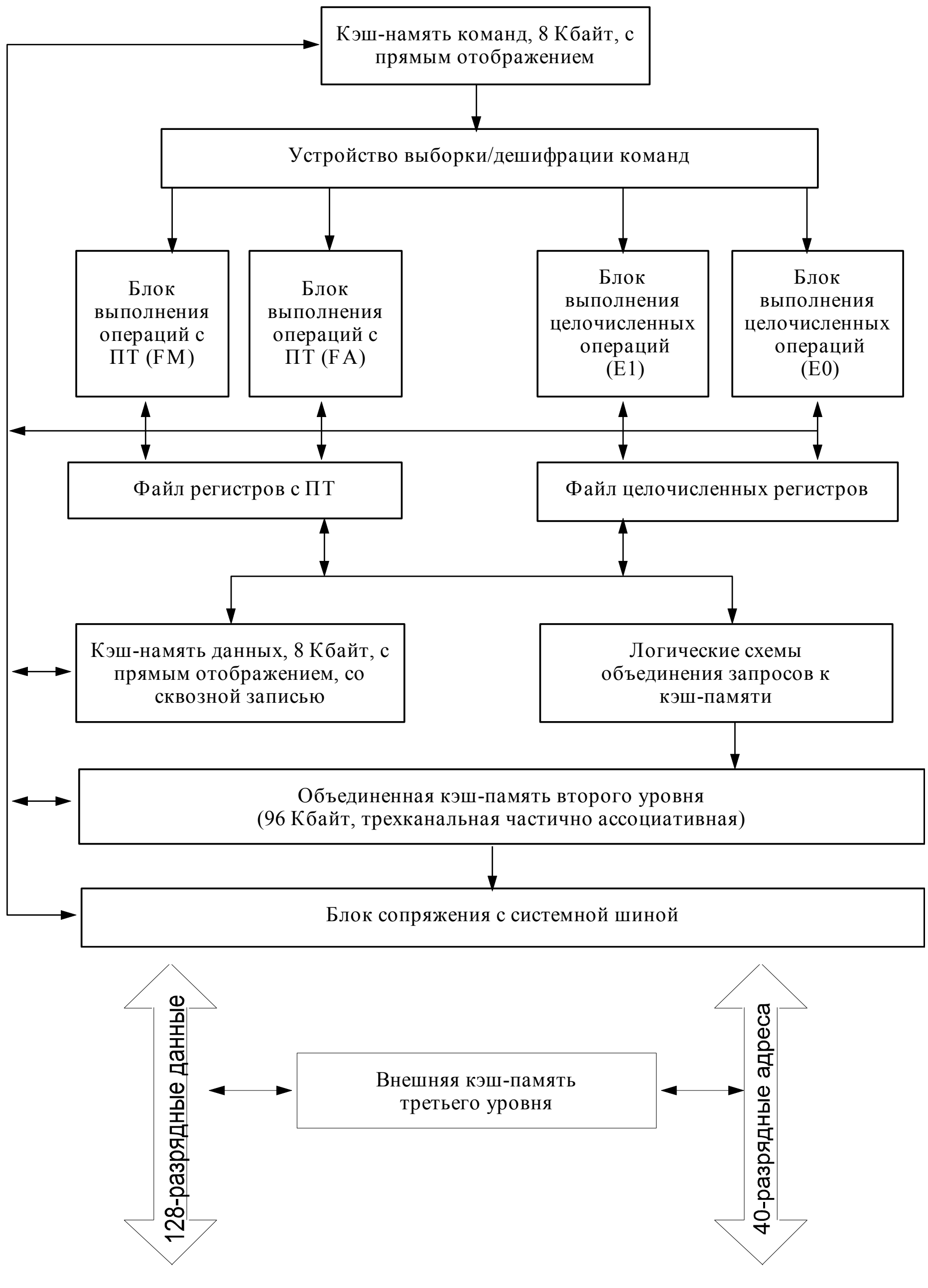


Рис. 8.19. Блок-схема процессора Alpha 21164

Устройство плавающей точки состоит из двух конвейерных исполнительных устройств: конвейера сложения, который выполняет все команды плавающей точки, за исключением команд умножения, и конвейер умножения, который выполняет команды умножения с плавающей точкой. Два специальных конвейера загрузки и один конвейер записи данных позволяют командам загрузки/записи выполняться параллельно с выполнением операций с плавающей точкой. Аппаратно поддерживаются все режимы округления, предусмотренные стандартами IEEE и VAX.

Устройство управления памятью выполняет все команды загрузки, записи и барьерные операции синхронизации. Оно содержит полностью ассоциативный 64-строчный буфер преобразования адресов (DTB), 8 Кбайт кэш-память данных с прямым отображением, файл адресов промахов и буфер записи. Длина строки в кэше данных равна 32 байтам, он имеет два порта по чтению и реализован по принципу сквозной записи. Он индексируется разрядами физического адреса, а в тегах хранятся физические адреса. В устройство управления памятью в каждом такте может поступать до двух виртуальных адресов из целочисленного устройства. DTB также имеет два порта, поэтому он может одновременно выполнять преобразование двух виртуальных адресов в физические. Команды загрузки обращаются к кэшу данных и возвращают результат в регистровый файл в случае попадания. При этом задержка составляет два такта. В случае промаха физические адреса направляются в файл адресов промахов, где они буферизуются и ожидают завершения обращения к кэш-памяти второго уровня. Команды записи записывают данные в кэш данных в случае попадания и всегда помещают данные в буфер записи, где они ожидают обращения к кэш-памяти второго уровня.

Отличительной особенностью микропроцессора 21164 является размещение на кристалле вторичного трехканального множественно-ассоциативного кэша, емкостью 96 Кбайт. Вторичный кэш резко снижает количество обращений к внешней шине микропроцессора. Кроме вторичного кэша на кристалле поддерживается работа с внешним кэшем третьего уровня.

Сочетание большого количества вычислительных устройств, более быстрого выполнения операций с плавающей точкой (четыре такта вместо шести), более быстрого доступа к первичному кэшу (два такта вместо трех) обеспечивают новому микропроцессору рекордные параметры производительности.

8.6. 8.6. Особенности архитектуры POWER компании IBM и PowerPC компаний Motorola, Apple и IBM

Как уже было отмечено, одним из разработчиков фундаментальной концепции RISC-архитектуры был Джон Кук из Исследовательского центра IBM им. Уотсона, который в середине 70-х проводил исследования в этом направлении и построил миникомпьютер IBM 801, который так никогда и не появился на рынке. Дальнейшее развитие этих идей в компании IBM нашло отражение при разработке архитектуру POWER в конце 80-х. Архитектура POWER (и ее поднаправления POWER2 и PowerPC) в настоящее время являются основой семейства рабочих станций и серверов RISC System /6000 компании IBM.

Развитие архитектуры IBM 801 в направлении POWER шло в следующих направлениях: воплощение концепции суперскалярной обработки, улучшение архитектуры как целевого объекта компиляторов, сокращение длины конвейера и времени выполнения команд и, наконец, приоритетная ориентация на эффективное выполнение операций с плавающей точкой.

8.6.1. 8.6.1. Архитектура POWER

Архитектура POWER во многих отношениях представляет собой традиционную RISC-архитектуру. Она придерживается наиболее важных отличительных особенностей RISC: фиксированной длины команд, архитектуры регистр-регистр, простых способов адресации, простых (не требующих интерпретации) команд, большого регистрового файла и трехоперандного (неразрушительного) формата команд. Однако архитектура POWER имеет также несколько дополнительных свойств, которые отличают ее от других RISC-архитектур.

Во-первых, набор команд был основан на идее суперскалярной обработки. В базовой архитектуре команды распределяются по трем независимым исполнительным устройствам: устройству переходов, устройству с фиксированной точкой и устройству с плавающей точкой. Команды могут направляться в каждое из этих устройств одновременно, где они могут выполняться одновременно и заканчиваться не в порядке поступления. Для увеличения уровня параллелизма, который может быть достигнут на практике, архитектура набора команд определяет для каждого из устройств независимый набор регистров. Это минимизирует связи и синхронизацию, требуемые между устройствами, позволяя тем самым исполнительным устройствам настраиваться на динамическую смесь команд. Любая связь по данным, требующаяся между устройствами, должна анализироваться компилятором, который может ее эффективно спланировать. Следует отметить, что это только концептуальная модель. Любой конкретный процессор с архитектурой POWER может рассматривать любое из концептуальных устройств как множество исполнительных устройств для поддержки дополнительного параллелизма команд. Но существование модели приводит к согласованной разработке набора команд, который естественно поддерживает степень параллелизма по крайней мере равную трем.

Во-вторых, архитектура POWER расширена несколькими "смешанными" командами для сокращения времен выполнения. Возможно единственным недостатком технологии RISC по сравнению с CISC, является то, что иногда она использует большее количество команд для выполнения одного и того же задания. Было обнаружено, что во многих случаях увеличения размера кода можно избежать путем небольшого расширения набора команд, которое вовсе не означает возврат к сложным командам, подобным командам CISC. Например, значительная часть увеличения программного кода была обнаружена в кодах пролога и эпилога, связанных с сохранением и восстановлением регистров во время вызова процедуры. Чтобы устранить этот фактор IBM ввела команды "групповой загрузки и записи", которые обеспечивают пересылку нескольких регистров в/из памяти с помощью единственной команды. Соглашения о связях, используемые компиляторами POWER, рассматривают задачи планирования, разделяемые библиотеки и динамическое связывание как простой, единый механизм. Это было сделано с помощью косвенной адресации посредством таблицы содержания (TOC - Table Of Contents), которая модифицируется во время загрузки. Команды групповой загрузки и записи были важным элементом этих соглашений о связях.

Другим примером смешанных команд является возможность модификации базового регистра вновь вычисленным эффективным адресом при выполнении операций загрузки или записи (аналог автоинкрементной адресации). Эти команды устраняют необходимость выполнения дополнительных команд сложения, которые в противном случае потребовались бы для инкрементирования индекса при обращениях к массивам. Хотя это смешанная операция, она не мешает работе традиционного RISC-конвейера, поскольку модифицированный адрес уже вычислен и порт записи регистрового файла во время ожидания операции с памятью свободен.

Архитектура POWER обеспечивает также несколько других способов сокращения времени выполнения команд такие как: обширный набор команд для манипуляции битовыми полями, смешанные команды умножения-сложения с плавающей точкой, установку регистра условий в качестве побочного эффекта нормального выполнения команды и команды загрузки и записи строк (которые работают с произвольно выровненными строками байтов).

Третьим фактором, который отличает архитектуру POWER от многих других RISC-архитектур, является отсутствие механизма "задержанных переходов". Обычно этот механизм обеспечивает выполнение команды, следующей за командой условного перехода, перед выполнением самого перехода. Этот механизм эффективно работал в ранних RISC-машинах для заполнения "пузыря", появляющегося при оценке условий для выбора направления перехода и выборки нового потока команд. Однако в более продвинутых, суперскалярных машинах, этот механизм может оказаться неэффективным, поскольку один такт задержки команды перехода может привести к появлению нескольких "пузырей", которые не могут быть покрыты с помощью одного архитектурного слота задержки. Почти все такие машины, чтобы устранить влияние этих "пузырей", вынуждены вводить дополнительное оборудование (например, кэш-память адресов переходов). В таких машинах механизм задержанных переходов становится не только мало эффективным, но и привносит значительную сложность в логику обработки последовательности команд. Вместо этого архитектура переходов POWER была организована для поддержки методики "предварительного просмотра условных переходов" (branch-lockahead) и методики "свертывания переходов" (branch-folding).

Методика реализации условных переходов, используемая в архитектуре POWER, является четвертым уникальным свойством по сравнению с другими RISC-процессорами. Архитектура POWER определяет расширенные свойства регистра условий. Проблема архитектур с традиционным регистром условий заключается в том, что установка битов условий как побочного эффекта выполнения команды, ставит серьезные ограничения на возможность компилятора изменить порядок следования команд. Кроме того, регистр условий представляет собой единственный архитектурный ресурс, создающий серьезное узкое горло в машине, которая параллельно выполняет несколько команд или выполняет команды не в порядке их появления в программе. Некоторые RISC-архитектуры обходят эту проблему путем полного исключения из своего состава регистра условий и требуют установки кода условий с помощью команд сравнения в универсальный регистр, либо путем включения операции сравнения в саму команду перехода. Последний подход потенциально перегружает конвейер команд при выполнении перехода. Поэтому архитектура POWER вместо того, чтобы исправлять проблемы, связанные с традиционным подходом к регистру условий, предлагает: a) наличие специального бита в коде операции каждой команды, что делает модификацию регистра условий дополнительной возможностью, и тем самым восстанавливает способность компилятора реорганизовать код, и b) несколько (восемь) регистров условий для того, чтобы обойти проблему единственного ресурса и обеспечить большее число имен регистра условий так, что компилятор может разместить и распределить ресурсы регистра условий, как он это делает для универсальных регистров.

Другой причиной выбора модели расширенного регистра условий является то, что она согласуется с организацией машины в виде независимых исполнительных устройств. Концептуально регистр условий является локальным по отношению к устройству переходов. Следовательно, для оценки направления выполнения условного перехода не обязательно обращаться к универсальному регистровому файлу (который является локальным для устройства с фиксированной точкой). Для той степени, с которой компилятор может заранее спланировать модификацию кода условия (и/или загрузить заранее регистры адреса перехода), аппаратура может заранее просмотреть и свернуть условные переходы, выделяя их из потока команд. Это позволяет освободить в конвейере временной слот (такт) выдачи команды, обычно занятый командой перехода, и дает возможность диспетчеру команд создавать непрерывный линейный поток команд для вычислительных исполнительных устройств.

Первая реализация архитектуры POWER появилась на рынке в 1990 году. С тех пор компания IBM представила на рынок еще две версии процессоров POWER2 и POWER2+, обеспечивающих поддержку кэш-памяти второго уровня и имеющих расширенный набор команд.

По данным IBM процессор POWER требует менее одного такта для выполнении одной команды по сравнению с примерно 1.25 такта у процессора Motorola 68040, 1.45 такта у процессора SPARC, 1.8 такта у Intel i486DX и 1.8 такта Hewlett-Packard PA-RISC. Тактовая частота архитектурного ряда в зависимости от модели меняется от 25 МГц до 62 МГц.

Процессоры POWER работают на частоте 33, 41.6, 45, 50 и 62.5 МГЦ. Архитектура POWER включает раздельную кэш-память команд и данных (за исключением рабочих станций и серверов рабочих групп начального уровня, которые имеют однокристальную реализацию процессора POWER и общую кэш-память команд и данных), 64- или 128-битовую шину памяти и 52-битовый виртуальный адрес. Она также имеет интегрированный процессор плавающей точки и таким образом хорошо подходит для приложений с интенсивными вычислениями, типичными для технической среды, хотя текущая стратегия RS/6000 нацелена как на коммерческие, так и на технические приложения. RS/6000 показывает хорошую производительность на плавающей точке: 134.6 SPECp92 для POWERstation/Powerserver 580. Это меньше, чем уровень моделей Hewlett-Packard 9000 Series 800 G/H/I-50, которые достигают уровня 150 SPECfp92.

Для реализации быстрой обработки ввода/вывода в архитектуре POWER используется шина Micro Channel, имеющая пропускную способность 40 или 80 Мбайт/сек. Шина Micro Channel включает 64-битовую шину данных и обеспечивает поддержку работы нескольких главных адаптеров шины. Такая поддержка позволяет сетевым контроллерам, видеоадаптерам и другим интеллектуальным устройствам передавать информацию по шине независимо от основного процессора, что снижает нагрузку на процессор и соответственно увеличивает системную производительность.

Многокристальный набор POWER2 состоит из восьми полузаказных микросхем (устройств):

- • Блок кэш-памяти команд (ICU) - 32 Кбайт, имеет два порта с 128-битовыми шинами;
- • Блок устройств целочисленной арифметики (FXU) - содержит два целочисленных конвейера и два блока регистров общего назначения (по 32 32-битовых регистра). Выполняет все целочисленные и логические операции, а также все операции обращения к памяти;
- • Блок устройств плавающей точки (FPU) - содержит два конвейера для выполнения операций с плавающей точкой двойной точности, а также 54 64-битовых регистра плавающей точки;
- • Четыре блока кэш-памяти данных - максимальный объем кэш-памяти первого уровня составляет 256 Кбайт. Каждый блок имеет два порта. Устройство реализует также ряд функций обнаружения и коррекции ошибок при взаимодействии с системой памяти;
- • Блок управления памятью (MMU).

Набор кристаллов POWER2 содержит порядка 23 миллионов транзисторов на площади 1217 квадратных мм и изготовлен по технологии КМОП с проектными нормами 0.45 микрон. Рассеиваемая мощность на частоте 66.5 МГц составляет 65 Вт.

Производительность процессора POWER2 по сравнению с POWER значительно повышена: при тактовой частоте 71.5 МГц она достигает 131 SPECint92 и 274 SPECfp92.

8.6.2. 8.6.2. Эволюция архитектуры POWER в направлении архитектуры PowerPC

Компания IBM распространяет влияние архитектуры POWER в направлении малых систем с помощью платформы PowerPC. Архитектура POWER в этой форме может обеспечивать уровень производительности и масштабируемость, превышающие возможности современных персональных компьютеров. PowerPC базируется на платформе RS/6000 в дешевой конфигурации. В архитектурном плане основные отличия этих двух разработок заключаются лишь в том, что системы PowerPC используют однокристальную реализацию архитектуры POWER, изготавливаемую компанией Motorola, в то время как большинство систем RS/6000 используют многокристальную реализацию. Имеется несколько вариаций процессора PowerPC, обеспечивающих потребности портативных изделий и настольных рабочих станций, но это не исключает возможность применения этих процессоров в больших системах. Первым на рынке был объявлен процессор 601, предназначенный для использования в настольных рабочих станциях компаний IBM и Apple. За ним последовали кристаллы 603 для портативных и настольных систем начального уровня и 604 для высокопроизводительных настольных систем. Наконец, процессор 620 разработан специально для серверных конфигураций и ожидается, что со своей 64-битовой организацией он обеспечит исключительно высокий уровень производительности.
При разработке архитектуры PowerPC для удовлетворения потребностей трех различных компаний (Apple, IBM и Motorola) при сохранении совместимости с RS/6000, в архитектуре POWER было сделано несколько изменений в следующих направлениях:

- • упрощение архитектуры с целью ее приспособления ее для реализации дешевых однокристальных процессоров;
- • устранение команд, которые могут стать препятствием повышения тактовой частоты;
- • устранение архитектурных препятствий суперскалярной обработке и внеочередному выполнению команд;
- • добавление свойств, необходимых для поддержки симметричной многопроцессорной обработки;
- • добавление новых свойств, считающихся необходимыми для будущих прикладных программ;
- • ясное определение линии раздела между "архитектурой" и "реализацией";
- • обеспечение длительного времени жизни архитектуры путем ее расширения до 64-битовой.

Архитектура PowerPC поддерживает ту же самую базовую модель программирования и назначение кодов операций команд, что и архитектура POWER. В тех местах, где были сделаны изменения, которые могли потенциально препятствовать процессорам PowerPC выполнять существующие двоичные коды RS/6000, были расставлены "ловушки", обеспечивающие прерывание и эмуляцию с помощью программного обеспечения. Такие изменения вводились, естественно, только в тех случаях, если соответствующая возможность либо использовалась не очень часто в кодах прикладных программ, либо была изолирована в библиотечных программах, которые можно просто заменить.

PowerPC 601

Первый микропроцессор PowerPC, PowerPC 601, в настоящее время выпускается как компанией IBM, так и компанией Motorola. Он представляет собой процессор среднего класса и предназначен для использования в настольных вычислительных системах малой и средней стоимости. Он был разработан в качестве переходной модели от архитектуры POWER к архитектуре PowerPC и реализует возможности обеих архитектур. При этом двоичные коды RS/6000 выполняются на нем без изменений, что дало дополнительное время разработчикам компиляторов для освоения архитектуры PowerPC, а также разработчикам прикладных систем, которые должны перекомпилировать свои программы, чтобы полностью использовать возможности архитектуры PowerPC.

Процессор 601 базировался на однокристальном процессоре IBM, который был разработан к моменту создания альянса трех ведущих фирм. Но по сравнению со своим предшественником, PowerPC 601 претерпел серьезные изменения в сторону повышения производительности и снижения стоимости. Например, в его состав было включено более сложное устройство переходов, расширенные возможностями мультипроцессорной работы, включая интерфейс шины высокопроизводительного процессора 88110 компании Motorola. В Power 601 реализована суперскалярная обработка, позволяющая выдавать на выполнение в каждом такте 3 команды, возможно не в порядке их расположения в программном коде.

Процессор PowerPC 603

PowerPC 603 является первым микропроцессором в семействе PowerPC, который полностью поддерживает архитектуру PowerPC (рисунок 8.20). Он включает пять функциональных устройств: устройство переходов, целочисленное устройство, устройство плавающей точки, устройство загрузки/записи и устройство системных регистров, а также две, расположенных на кристалле кэш-памяти для команд и данных, емкостью по 8 Кбайт. Поскольку PowerPC 603 - суперскалярный микропроцессор, он может выдавать в эти исполнительные устройства и завершать выполнение до трех команд в каждом такте. Для увеличения производительности PowerPC 603 допускает внеочередное выполнение команд. Кроме того, он обеспечивает программируемые режимы снижения потребляемой мощности, которые дают разработчикам систем гибкость реализации различных технологий управления питанием.

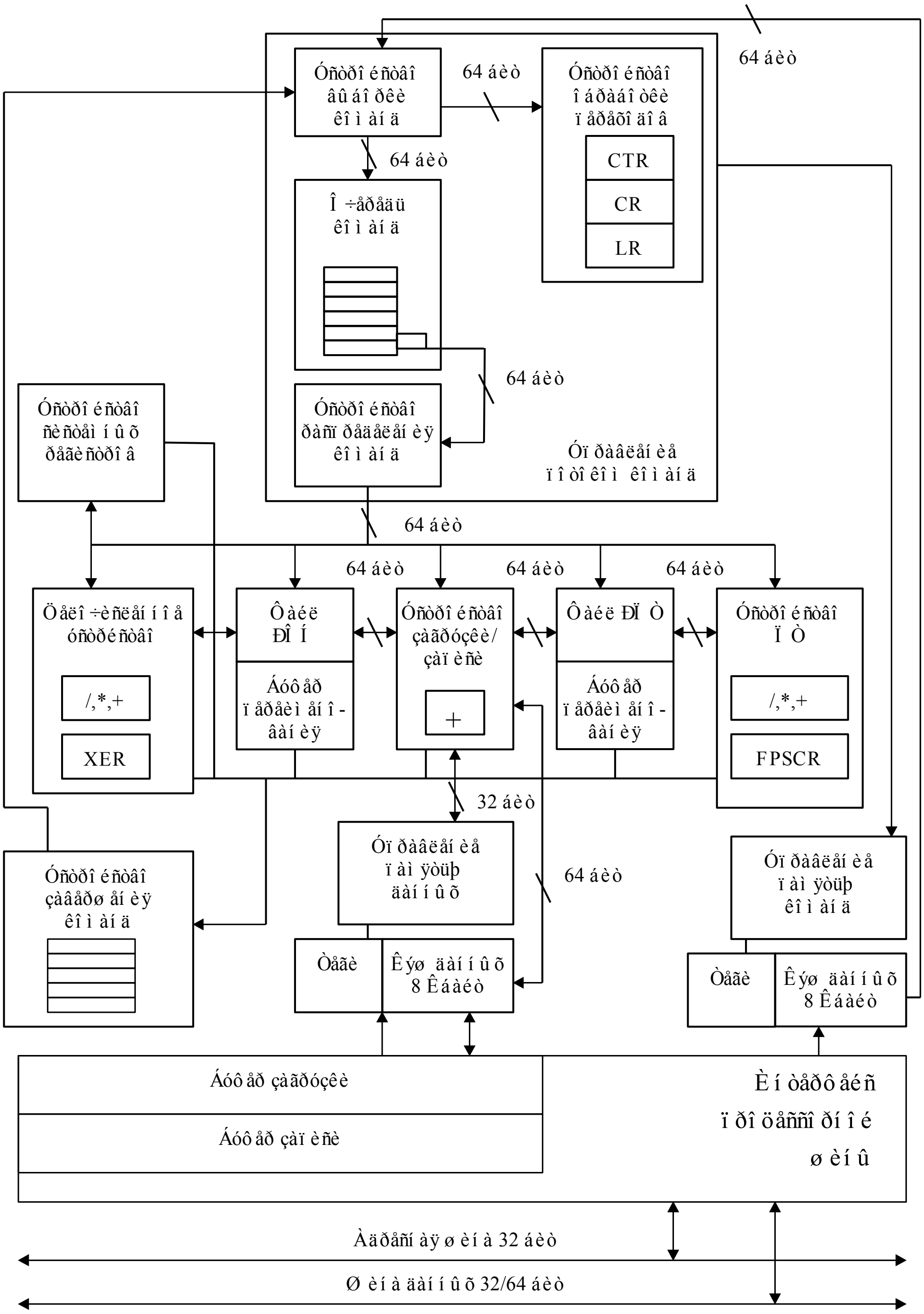


Рис. 8.20. Блок-схема процессора Power PC 603

При обработке в процессоре команды распределяются по пяти исполнительным устройствам в заданном программой порядке. Если отсутствуют зависимости по операндам, выполнение происходит немедленно. Целочисленное устройство выполняет большинство команд за один такт. Устройство плавающей точки имеет конвейерную организацию и выполняет операции с плавающей точкой, как с одинарной, так и с двойной точностью. Команды условных переходов обрабатывается в устройстве переходов. Если условия перехода доступны, то решение о направлении перехода принимается немедленно, в противном случае выполнение последующих команд

продолжается по предположению (спекулятивно). Команды, модифицирующие состояние регистров управления процессором, выполняются устройством системных регистров. Наконец, пересылки данных между кэш-памятью данных, с одной стороны, и регистрами общего назначения и регистрами плавающей точки, с другой стороны, обрабатываются устройством загрузки/записи.
В случае промаха при обращении к кэш-памяти, обращение к основной памяти осуществляется с помощью 64-битовой высокопроизводительной шины, подобной шине микропроцессора MC88110. Для максимизации пропускной способности и, как следствие, увеличения общей производительности кэш-память взаимодействует с основной памятью главным образом посредством групповых операций, которые позволяют заполнить строку кэш-памяти за одну транзакцию.

После окончания выполнения команды в исполнительном устройстве ее результаты направляются в буфер завершения команд (completion buffer) и затем последовательно записываются в соответствующий регистровый файл по мере изъятия команд из буфера завершения. Для минимизации конфликтов по регистрам, в процессоре PowerPC 603 предусмотрены отдельные наборы из 32 целочисленных регистров общего назначения и 32 регистров плавающей точки.

PowerPC 604

Суперскалярный процессор PowerPC 604 обеспечивает одновременную выдачу до четырех команд. При этом параллельно в каждом такте может завершаться выполнение до шести команд. На рисунке 8.21 представлена блок-схема процессора 604. Процессор включает шесть исполнительных устройств, которые могут работать параллельно:

- • устройство плавающей точки (FPU);
- • устройство выполнения переходов (BPU);
- • устройство загрузки/записи (LSU);
- • три целочисленных устройства (IU):
  - ◊ два однотактных целочисленных устройства (SCIU);
  - ◊ одно многотактное целочисленное устройство (MCIU).

Такая параллельная конструкция в сочетании со спецификацией команд PowerPC, допускающей реализацию ускоренного выполнения команд, обеспечивает высокую эффективность и большую пропускную способность процессора. Применяемые в процессоре 604 буфера переименования регистров, буферные станции резервирования, динамическое прогнозирование направления условных переходов и устройство завершения выполнения команд существенно увеличивают пропускную способность системы, гарантируют завершение выполнения команд в порядке, предписанном программой, и обеспечивают реализацию модели точного прерывания.

В процессоре 604 имеются отдельные устройства управления памятью и отдельные по 16 Кбайт внутренние кэши для команд и данных. В нем реализованы два буфера преобразования виртуальных адресов в физические TLB (отдельно для команд и для данных), содержащие по 128 строк. Оба буфера являются двухканальными множественно-ассоциативными и обеспечивают переменный размер страниц виртуальной памяти. Кэш-памяти и буфера TLB используют для замещения блоков алгоритм LRU.

Процессор 604 имеет 64-битовую внешнюю шину данных и 32-битовую шину адреса. Интерфейсный протокол процессора 604 позволяет нескольким главным устройствам шины конкурировать за системные ресурсы при наличии централизованного внешнего арбитра. Кроме того, внутренние логические схемы наблюдения за шиной поддерживают когерентность кэш-памяти в мультипроцессорных конфигурациях.

Процессор 604 обеспечивает как одиночные, так и групповые пересылки данных при обращении к основной памяти.

PowerPC 620

К концу 1995 года ожидается появление нового процессора PowerPC 620. В отличие от своих предшественников это будет полностью 64-битовый процессор. При работе на тактовой частоте 133 МГц его производительность оценивается в 225 единиц SPECint92 и 300 единиц SPECfp92, что соответственно на 40 и 100% больше показателей процессора PowerPC 604.

Подобно другим 64-битовым процессорам, PowerPC 620 содержит 64-битовые регистры общего назначения и плавающей точки и обеспечивает формирование 64-битовых виртуальных адресов. При этом сохраняется совместимость с 32-битовым режимом работы, реализованным в других моделях семейства PowerPC.

В процессоре имеется кэш-память данных и команд общей емкостью 64 Кбайт, интерфейсные схемы управления кэш-памятью второго уровня, 128-битовая шина данных между процессором и основной памятью, а также логические схемы поддержания когерентного состояния памяти при организации многопроцессорной системы.

Процессор PowerPC 620 нацелен на рынок высокопроизводительных рабочих станций и серверов.

В заключении отметим, что в иллюстрациях к курсу приведены основные характеристики некоторых современных систем, построенных на рассмотренных в данном разделе процессорах.

9. 9. Организация ввода/вывода

9.1. 9.1. Введение

Вопросы организации ввода/вывода в вычислительной системе иногда оказываются вне внимания потребителей. Это привело к тому, что при оценке производительности системы часто используются только оценки производительности процессора, а оценкой системы ввода/вывода пренебрегают. Такое отношение к системам ввода/вывода, как к некоторым не очень важным понятиям, проистекает также из термина "периферия", который применяется к устройствам ввода/вывода.

Однако это противоречит здравому смыслу. Компьютер без устройств ввода/вывода - как автомобиль без колес - на таком автомобиле далеко не уедешь. Очевидно, одной из наиболее правильных оценок производительности системы является время ответа (время между моментом ввода пользователем задания и получения им результата), которое учитывает все накладные расходы, связанные с выполнением задания в системе, включая ввод/вывод.

Кроме того, важность системы ввода/вывода определяется еще и тем, что быстрое увеличение производительности процессоров настолько изменило принципы классификации компьютеров, что именно по организации ввода/вывода мы можем как-то грубо их отличать: разница между мейнфреймом и миникомпьютером заключается в том, что мейнфрейм может поддерживать намного больше терминалов и дисков; разница между миникомпьютером и рабочей станцией заключается в том, что рабочая станция имеет экран, клавиатуру и мышь; разница между файл-сервером и рабочей станцией заключается в том, что файл-сервер имеет диски и ленточные устройства, а экран, клавиатура и мышь отсутствуют; разница между рабочей станцией и персональным компьютером заключается лишь в том, что рабочие станции всегда соединены друг с другом с помощью локальной сети.

Уже сейчас мы можем наблюдать, что в компьютерах различного ценового класса от рабочих станций до суперкомпьютеров (суперсерверов) используется один и тот же тип микропроцессора. Различия в стоимости и производительности определяются практически только организацией систем памяти и ввода/вывода (а также количеством процессоров).

Как уже отмечалось, производительность процессоров растет со скоростью 50-100% в год. Если одновременно не улучшались бы характеристики систем ввод/вывода, то, очевидно, разработка новых систем зашла бы в тупик. Важность оценки работы систем ввода/вывода была осознана многими пользователями компьютеров. Были разработаны специальные тестовые программы, позволяющие оценить эффективность систем ввода/вывода. В частности, такие тесты применяются для оценки суперкомпьютеров, систем обработки транзакций и файл-серверов.

9.2. 9.2. Системные и локальные шины

В вычислительной системе, состоящей из множества подсистем, необходим механизм для их взаимодействия. Эти подсистемы должны быстро и эффективно обмениваться данными. Например, процессор, с одной стороны, должен быть связан с памятью, с другой стороны, необходима связь процессора с устройствами ввода/вывода. Одним из простейших механизмов, позволяющих организовать взаимодействие различных подсистем, является единственная центральная шина, к которой подсоединяются все подсистемы. Доступ к такой шине разделяется между всеми подсистемами. Подобная организация имеет два основных преимущества: низкая стоимость и универсальность.

Поскольку такая шина является единственным местом подсоединения для разных устройств, новые устройства могут быть легко добавлены, и одни и те же периферийные устройства можно даже применять в разных вычислительных системах, использующих однотипную шину. Стоимость такой организации получается достаточно низкой, поскольку для реализации множества путей передачи информации используется единственный набор линий шины, разделяемый множеством устройств.

Главным недостатком организации с единственной шиной является то, что шина создает узкое горло, ограничивая, возможно, максимальную пропускную способность ввода/вывода. Если весь поток ввода/вывода должен проходить через центральную шину, такое ограничение пропускной способности весьма реально. В коммерческих системах, где ввод/вывод осуществляется очень часто, а также в суперкомпьютерах, где необходимые скорости ввода/вывода очень высоки из-за высокой производительности процессора, одним из главных вопросов разработки является создание системы нескольких шин, способной удовлетворить все запросы.

Одна из причин больших трудностей, возникающих при разработке шин, заключается в том, что максимальная скорость шины главным образом лимитируется физическими факторами: длиной шины и количеством подсоединяемых устройств (и, следовательно, нагрузкой на шину). Эти физические ограничения не позволяют произвольно ускорять шины. Требования быстродействия (малой задержки) системы ввода/вывода и высокой пропускной способности являются противоречивыми. В современных крупных системах используется целый комплекс взаимосвязанных шин, каждая из которых обеспечивает упрощение взаимодействия различных подсистем, высокую пропускную способность, избыточность (для увеличения отказоустойчивости) и эффективность.

Традиционно шины делятся на шины, обеспечивающие организацию связи процессора с памятью, и шины ввода/вывода. Шины ввода/вывода могут иметь большую протяженность, поддерживать подсоединение многих типов устройств, и обычно следуют одному из шинных стандартов. Шины процессор-память, с другой стороны, сравнительно короткие, обычно высокоскоростные и соответствуют организации системы памяти для обеспечения максимальной пропускной способности канала память-процессор. На этапе разработки системы, для шины процессор-память заранее известны все типы и параметры устройств, которые должны соединяться между собой, в то время как разработчик шины ввода/вывода должен иметь дело с устройствами, различающимися по задержке и пропускной способности.

Как уже было отмечено, с целью снижения стоимости некоторые компьютеры имеют единственную шину для памяти и устройств ввода/вывода. Такая шина часто называется системной. Персональные компьютеры, как правило, строятся на основе одной системной шины в стандартах ISA, EISA или MCA. Необходимость сохранения баланса производительности по мере роста быстродействия микропроцессоров привела к двухуровневой организации шин в персональных компьютерах на основе локальной шины. Локальной шиной называется шина, электрически выходящая непосредственно на контакты микропроцессора. Она обычно объединяет процессор, память, схемы буферизации для системной шины и ее контроллер, а также некоторые вспомогательные схемы. Типичными примерами локальных шин являются VL-Bus и PCI.

Рассмотрим типичную транзакцию на шине. Шинная транзакция включает в себя две части: посылку адреса и прием (или посылку) данных. Шинные транзакции обычно определяются характером взаимодействия с памятью: транзакция типа "Чтение" передает данные из памяти (либо в ЦП, либо в устройство ввода/вывода), транзакция типа "Запись" записывает данные в память. В транзакции типа "Чтение" по шине сначала посылается в память адрес вместе с соответствующими сигналами управления, индицирующими чтение. Память отвечает, возвращая на шину данные с соответствующими сигналами управления. Транзакция типа "Запись" требует, чтобы ЦП или устройство в/в послало в память адрес и данные и не ожидает возврата данных.

Обычно ЦП вынужден простаивать во время интервала между посылкой адреса и получением данных при выполнении чтения, но часто он не ожидает завершения операции при записи данных в память.
Разработка шины связана с реализацией ряда дополнительных возможностей (рис. 9.1). Решение о выборе той или иной возможности зависит от целевых параметров стоимости и производительности. Первые три возможности являются очевидными: раздельные линии адреса и данных, более широкие (имеющие большую разрядность) шины данных и режим групповых пересылок (пересылки нескольких слов) дают увеличение производительности за счет увеличения стоимости.
Следующий термин, указанный в таблице, - количество главных устройств шины (bus master). Главное устройство шины - это устройство, которое может инициировать транзакции чтения или записи. ЦП, например, всегда является главным устройством шины. Шина имеет несколько главных устройств, если имеется несколько ЦП или когда устройства ввода/вывода могут инициировать транзакции на шине. Если имеется несколько таких устройств, то требуется схема арбитража, чтобы решить, кто следующий захватит шину. Арбитраж часто основан либо на схеме с фиксированным приоритетом, либо на более "справедливой" схеме, которая случайным образом выбирает, какое главное устройство захватит шину.
В настоящее время используются два типа шин, отличающиеся способом коммутации: шины с коммутацией цепей (circuit-switched bus) и шины с коммутацией пакетов (packet-switched bus), получившие свои названия по аналогии со способами коммутации в сетях передачи данных. Шина с коммутацией пакетов при наличии нескольких главных устройств шины обеспечивает значительно большую пропускную способность по сравнению с шиной с коммутацией цепей за счет разделения транзакции на две логические части: запроса шины и ответа. Такая методика получила название "расщепления" транзакций (split transaction). В некоторых системах такая возможность называется шиной соединения/разъединения (connect/disconnect) или конвейерной шиной (pipelined bus). Транзакция чтения разбивается на транзакцию запроса чтения, которая содержит адрес, и транзакцию ответа памяти, которая содержит данные. Каждая транзакция теперь должна быть помечена (тегирована) соответствующим образом, чтобы ЦП и память могли сообщить, что есть что.
Шина с коммутацией цепей не делает расщепления транзакций, любая транзакция на ней есть неделимая операция. Главное устройство запрашивает шину, после арбитража помещает на нее адрес и блокирует шину до окончания обслуживания запроса. Большая часть этого времени обслуживания при этом тратится не на выполнение операций на шине (например, на задержку выборки из памяти). Таким образом, в шинах с коммутацией цепей это время просто теряется. Расщепленные транзакции делают шину доступной для других главных устройств, пока память читает слово по запрошенному адресу. Это, правда, также означает, что ЦП должен бороться за шину для посылки данных, а память должна бороться за шину, чтобы вернуть данные. Таким образом, шина с расщеплением транзакций имеет более высокую пропускную способность, но обычно она имеет и большую задержку, чем шина, которая захватывается на все время выполнения транзакции. Транзакция называется расщепленной, поскольку произвольное количество других пакетов или транзакций могут использовать шину между запросом и ответом.
Последний вопрос связан с выбором типа синхронизации и определяет, является ли шина синхронной или асинхронной. Если шина синхронная, то она включает сигналы синхронизации, которые передаются по линиям управления шины, и фиксированный протокол, определяющий расположение сигналов адреса и данных относительно сигналов синхронизации. Поскольку практически никакой дополнительной логики не требуется для того, чтобы решить, что делать в следующий момент времени, эти шины могут быть и быстрыми, и дешевыми. Однако они имеют два главных недостатка. Все на шине должно происходить с одной и той же частотой синхронизации, поэтому из-за

проблемы перекоса синхросигналов, синхронные шины не могут быть длинными. Обычно шины процессор-память синхронные.

Асинхронная шина, с другой стороны, не тактируется. Вместо этого обычно используется старт-стопный режим передачи и протокол "квитирования" (handshaking) между источником и приемником данных на шине. Эта схема позволяет гораздо проще приспособить широкое разнообразие устройств и удлинить шину без беспокойства о перекосе сигналов синхронизации и о системе синхронизации. Если может использоваться синхронная шина, то она обычно быстрее, чем асинхронная, из-за отсутствия накладных расходов на синхронизацию шины для каждой транзакции. Выбор типа шины (синхронной или асинхронной) определяет не только пропускную способность, но также непосредственно влияет на емкость системы ввода/вывода в терминах физического расстояния и количества устройств, которые могут быть подсоединены к шине. Асинхронные шины по мере изменения технологии лучше масштабируются. Шины ввода/вывода обычно асинхронные.

Возможность	Высокая производительность	Низкая стоимость
Общая разрядность шины	Отдельные линии адреса и данных	Мультиплексирование линий адреса и данных
Ширина (разрядность) данных	Чем шире, тем быстрее (например, 32 бит)	Чем уже, тем дешевле (например, 8 бит)
Размер пересылки	Пересылка нескольких слов имеет меньшие накладные расходы	Пересылка одного слова дешевле
Главные устройства шины	Несколько (требуется арбитраж)	Одно (арбитраж не нужен)
Расщепленные транзакции?	Да - отдельные пакеты Запроса и Ответа дают большую полосу пропускания (нужно несколько главных устройств)	Нет - продолжающееся соединение дешевле и имеет меньшую задержку
Тип синхронизации	Синхронные	Асинхронные

Рис. 9.1. Основные возможности шин

Стандарты шин

Обычно количество и типы устройств ввода/вывода в вычислительных системах не фиксируются, что позволяет пользователю самому подобрать необходимую конфигурацию. Шина ввода/вывода компьютера может рассматриваться как шина расширения, обеспечивающая постепенное наращивание устройств ввода/вывода. Поэтому стандарты играют огромную роль, позволяя разработчикам компьютеров и устройств ввода/вывода работать независимо. Появление стандартов определяется разными обстоятельствами.

Иногда широкое распространение и популярность конкретных машин становятся причиной того, что их шина ввода/вывода становится стандартом де-факто. Примерами таких шин могут служить PDP-11 Unibus и IBM PC-AT Bus. Иногда стандарты появляются также в результате определенных достижений по стандартизации в некотором секторе рынка устройств ввода/вывода. Интеллектуальный периферийный интерфейс (IPI - Intelligent Peripheral Interface) и Ethernet являются примерами стандартов, появившихся в результате кооперации производителей. Успех того или иного стандарта в значительной степени определяется его принятием такими организациями как ANSI (Национальный институт по стандартизации США) или IEEE (Институт инженеров по электротехнике и радиоэлектронике). Иногда стандарт шины может быть прямо разработан одним из комитетов по стандартизации: примером такого стандарта шины является FutureBus.

На рис. 9.2 представлены характеристики нескольких стандартных шин. Заметим, что строки этой таблицы, касающиеся пропускной способности, не указаны в виде одной цифры для шин процессор-память (VME, FutureBus, MultibusII). Размер пересылки, из-за разных накладных расходов шины, сильно влияет на пропускную способность. Поскольку подобные шины обычно обеспечивают связь с памятью, то пропускная способность шины зависит также от быстродействия памяти. Например, в идеальном случае при бесконечном размере пересылки и бесконечно быстрой памяти (время доступа 0 нсек) шина FutureBus на 240% быстрее шины VME, но при пересылке одиночных слов из 150-нсекундной памяти шина FutureBus только примерно на 20% быстрее, чем шина VME.

	VME bus	FutureBus	Multibus II	IPI	SCSI
Ширина шины (кол-во сигналов)	128	96	96	16	8
Мультиплекси-рование адреса/данных	Нет	Да	Да	–	–
Разрядность данных	16/32 бит	32 бит	32 бит	16 бит	8 бит
Размер пересылки (слов)	Одиночная или групповая	Одиночная или групповая	Одиночная или групповая	Одиночная или групповая	Одиночная или групповая
Количество главных устройств шины	Несколько	Несколько	Несколько	Одно	Несколько
Расщепление транзакций	Нет	Доп. возможность	Доп. возможность	Доп. возможность	Доп. возможность
Полоса пропускания (время доступа - 0 нс - 1 слово)	25.9 Мб/с	37.0 Мб/с	20.0 Мб/с	25.0 Мб/с	5.0 Мб/с
Полоса пропускания (время доступа - 150 нс - 1 слово)	12.9 Мб/с	15.5 Мб/с	10.0 Мб/с	25.0 Мб/с	5.0 Мб/с
Полоса пропускания (время доступа - 0 нс - неогр. размер блока)	27.9 Мб/с	95.2 Мб/с	40.0 Мб/с	25.0 Мб/с	5.0 Мб/с
Полоса пропускания (время доступа - 150 нс - неогр. размер блока)	13.6 Мб/с	20.8 Мб/с	13.3 Мб/с	25.0 Мб/с	5.0 Мб/с
Максимальное количество устройств	21	20	21	8	7
Максимальная длина шины	0.5 м	0.5 м	0.5 м	50 м	25 м
Стандарт	IEEE 1014	IEEE 896.1	ANSI/ IEEE 1296	ANSI X3.129	ANSI X3.131

Рис. 9.2. Примеры стандартных шин

Одной из популярных шин персональных компьютеров была системная шина IBM PC/XT, обеспечивавшая передачу 8 бит данных. Кроме того, эта шина включала 20 адресных линий, которые ограничивали адресное пространство пределом в 1 Мбайт. Для работы с внешними устройствами в этой шине были предусмотрены также 4 линии аппаратных прерываний (IRQ) и 4 линии для требования внешними устройствами прямого доступа к памяти (DMA). Для подключения плат расширения использовались специальные 62-контактные разъемы. При этом системная шина и микропроцессор синхронизировались от одного тактового генератора с частотой 4.77 МГц. Таким образом теоретическая скорость передачи данных могла достигать немногим более 4 Мбайт/с.

Системная шина *ISA (Industry Standard Architecture)* впервые стала применяться в персональных компьютерах IBM PC/AT на базе процессора i286. Эта системная шина отличалась наличием второго, 36-контактного дополнительного разъема для соответствующих плат расширения. За счет этого количество адресных линий было увеличено на 4, а данных - на 8, что позволило передавать параллельно 16 бит данных и обращаться к 16 Мбайт системной памяти. Количество линий аппаратных прерываний в этой шине было увеличено до 15, а каналов прямого доступа - до 7. Системная шина ISA полностью включала в себя возможности старой 8-разрядной шины. Шина ISA позволяет синхронизировать работу процессора и шины с разными тактовыми частотами. Она работает на частоте 8 МГц, что соответствует максимальной скорости передачи 16 Мбайт/с.

С появлением процессоров i386, i486 и Pentium шина ISA стала узким местом персональных компьютеров на их основе. Новая системная шина *EISA (Extended Industry Standard Architecture),* появившаяся в конце 1988 года, обеспечивает адресное пространство в 4 Гбайта, 32-битовую передачу данных (в том числе и в режиме DMA), улучшенную систему прерываний и арбитраж DMA, автоматическую конфигурацию системы и плат расширения. Устройства шины ISA могут работать на шине EISA. Шина EISA предусматривает централизованное управление доступом к шине за счет наличия специального устройства - арбитра шины. Поэтому к ней может подключаться несколько главных устройств шины. Улучшенная система прерываний позволяет подключать к каждой физической линии запроса на прерывание несколько устройств, что снимает проблему количества линий прерывания. Шина EISA тактируется частотой около 8 МГц и имеет максимальную теоретическую скорость передачи данных 33 Мбайт/с.

Шина MCA также обеспечивает 32-разрядную передачу данных, тактируется частотой 10 МГц, имеет средства автоматического конфигурирования и арбитража запросов. В отличие от EISA она не совместима с шиной ISA и используется только в компьютерах компании IBM.

Шина VL-bus, предложенная ассоциацией VESA (Video Electronics Standard Association), предназначалась для увеличения быстродействия видеоадаптеров и контроллеров дисковых накопителей для того, чтобы они могли работать с тактовой частотой до 40 МГц. Шина VL-bus имеет 32 линии данных и позволяет подключать до трех периферийных устройств, в качестве которых наряду с видеоадаптерами и дисковыми контроллерами могут выступать и сетевые адаптеры. Максимальная скорость передачи данных по шине VL-bus может составлять около 130 Мбайт/с. После появления процессора Pentium ассоциация VESA приступила к работе над новым стандартом VL-bus версии 2, который предусматривает использование 64-битовой шины данных и увеличение количества разъемов расширения. Ожидаемая скорость передачи данных - до 400 Мбайт/с.

Шина PCI (Peripheral Component Interconnect) также, как и шина VL-bus, поддерживает 32-битовый канал передачи данных между процессором и периферийными устройствами. Она работает на тактовой частоте 33 МГц и имеет максимальную пропускную способность 120 Мбайт/с. При работе с процессорами i486 шина PCI дает примерно те же показатели производительности, что и шина VL-bus. Однако, в отличие от последней, шина PCI является процессорно-независимой (шина VL-bus подключается непосредственно к процессору i486 и только к нему). Ее легко подключить к различным центральным процессорам. В их числе Pentium, Alpha, R4400 и PowerPC.

Шина VME приобрела большую популярность как шина ввода/вывода в рабочих станциях и серверах на базе RISC-процессоров. Эта шина высоко стандартизована, имеется несколько версий этого стандарта. В частности, VME32 - 32-битовая шина с производительностью 30 Мбайт/с, а VME64 - 64-битовая шина с производительностью 160 Мбайт/с.

В однопроцессорных и многопроцессорных рабочих станциях и серверах на основе микропроцессоров SPARC одновременно используются несколько типов шин: SBus, MBus и XDBus, причем шина SBus применяется в качестве шины ввода/вывода, а MBus и XDBus - в качестве шин для объединения большого числа процессоров и памяти.

Шина SBus (известная также как стандарт IEEE-1496) имеет 32-битовую и 64-битовую реализацию, работает на частоте 20 и 25 МГц и имеет максимальную скорость передачи данных в 32-битовом режиме равную соответственно 80 или 100 Мбайт/с. Шина предусматривает режим групповой пересылки данных с максимальным размером пересылки до 128 байт. Она может работать в двух режимах передачи данных: режиме программируемого ввода/вывода и в режиме прямого доступа к виртуальной памяти (DVMA). Последний режим особенно эффективен при передаче больших блоков данных.

Шина MBus работает на тактовой частоте 50 МГц в синхронном режиме с мультиплексированием адреса и данных. Общее число сигналов шины равно 100, а разрядность шины данных составляет 64 бит. По шине передаются 36-битовые физические адреса. Шина обеспечивает протокол поддержания когерентного состояния кэш-памяти нескольких (до четырех) процессоров. Она имеет максимальную пропускную способность в 400 Мбайт/с, а типовая скорость передачи составляет 125 Мбайт/с. Отличительными свойствами шины MBus являются: возможность увеличения числа процессорных модулей, поддержка симметричной мультипроцессорной обработки, высокая пропускная способность при обмене с памятью и подсистемой ввода/вывода, открытые (непатентованные) спецификации интерфейсов.

Шина MBus была разработана для относительно небольших систем (ее длина ограничивается десятью дюймами, что позволяет объединить до четырех процессоров с кэш-памятью второго уровня и основной памятью). Для построения систем с большим числом процессоров нужна большая масштабируемость шины. Одна из подобного рода шин - XDBus, используется в серверах SPARCserver 1000 (до 8 процессоров) и SPARCcenter 2000 (до 20 процессоров) компании Sun Microsystems и SuperServer 6400 компании Cray Research (до 64 процессоров). XDBus представляет собой шину, работающую в режиме расщепления транзакций. Это позволяет ей, имея пиковую производительность в 400 Мбайт/с, поддерживать типовую скорость передачи на уровне более 310 Мбайт/с.

В современных компьютерах часто применяются и фирменные (запатентованные) шины, обеспечивающие очень высокую пропускную способность для построения многопроцессорных серверов. Одной из подобных шин является системная шина POWERpath-2, которая применяется в суперсервере Chellenge компании Silicon Graphics. Она способна поддерживать эффективную работу до 36 процессоров MIPS R4400 (9 процессорных плат с четырьмя 150 МГц процессорами на каждой плате) с общей расслоенной памятью объемом до 16 Гбайт (коэффициент расслоения памяти равен восьми). POWERpath-2 имеет разрядность данных 256 бит, разрядность адреса 40 бит, и работает на частоте 50 МГц с пониженным напряжением питания. Она поддерживает методику расщепления транзакций, причем может иметь до восьми отложенных транзакций чтения одновременно. При этом арбитраж шины адреса и шины данных выполняется независимо. POWERpath-2 поддерживает протокол когерентного состояния кэш-памяти каждого процессора в системе.

Одной из наиболее популярных шин ввода-вывода в настоящее время является шина SCSI.

Под термином SCSI - Small Computer System Interface (Интерфейс малых вычислительных систем) обычно понимается набор стандартов, разработанных Национальным институтом стандартов США (ANSI) и определяющих механизм реализации магистрали передачи данных между системной шиной компьютера и периферийными устройствами. На сегодняшний день приняты два стандарта (SCSI-1 и SCSI-2). Стандарт SCSI-3 находится в процессе доработки.

Начальный стандарт 1986 года, известный теперь под названием SCSI-1, определял рабочие спецификации протокола шины, набор команд и электрические параметры. В 1992 году этот стандарт был пересмотрен с целью устранения недостатков первоначальной спецификации (особенно в части синхронного режима передачи данных) и добавления новых возможностей повышения производительности, таких как "быстрый режим" (fast mode), "широкий режим" (wide mode) и помеченные очереди. Этот пересмотренный стандарт получил название SCSI-2 и в настоящее время используется большинством поставщиков вычислительных систем.

Первоначально SCSI предназначался для использования в небольших дешевых системах и поэтому был ориентирован на достижение хороших результатов при низкой стоимости. Характерной его чертой является простота, особенно в части обеспечения гибкости конфигурирования периферийных устройств без изменения организации основного процессора. Главной особенностью подсистемы SCSI является размещение в периферийном оборудовании интеллектуального контроллера.

Для достижения требуемого высокого уровня независимости от типов периферийных устройств в операционной системе основной машины, устройства SCSI представляются имеющими очень простую архитектуру. Например, геометрия дискового накопителя представляется в виде линейной последовательности одинаковых блоков, хотя в действительности любой диск имеет более сложную многомерную геометрию, содержащую поверхности, цилиндры, дорожки, характеристики плотности, таблицу дефектных блоков и множество других деталей. В этом случае само устройство или его контроллер несут ответственность за преобразование упрощенной SCSI модели в данные для реального устройства.

Стандарт SCSI-2 определяет в частности различные режимы: Wide SCSI, Fast SCSI и Fast-and-Wide SCSI. Стандарт SCSI-1 определяет построение периферийной шины на основе 50-жильного экранированного кабеля, описывает методы адресации и электрические характеристики сигналов. Шина данных SCSI-1 имеет разрядность 8 бит, а максимальная скорость передачи составляет 5 Мбайт/сек. Fast SCSI сохраняет 8-битовую шину данных и тем самым может использовать те же самые физические кабели, что и SCSI-1. Он отличается только тем, что допускает передачи со скоростью 10 Мбайт/сек в синхронном режиме. Wide SCSI удваивает либо учетверяет разрядность шины данных (либо 16, либо 32 бит), допуская соответственно передачи со скоростью либо 10, либо 20 Мбайт/сек. В комбинации Fast-and-Wide SCSI возможно достижение скоростей передачи 20 и 40 Мбайт/сек соответственно.

Однако поскольку в обычном 50-жильном кабеле просто не хватает жил, комитет SCSI решил расширить спецификацию вторым 66-жильным кабелем (так называемый B-кабель). B-кабель имеет дополнительные линии данных и ряд других сигнальных линий, позволяющие реализовать режим Fast-and-Wide.

В реализации режима Wide SCSI предложена также расширенная адресация, допускающая подсоединение к шине до 16 устройств (вместо стандартных восьми). Это значительно увеличивает гибкость подсистемы SCSI, правда приводит к появлению дополнительных проблем, связанных с эффективностью ее использования.

Реализация режимов Wide-SCSI и Fast-and-Wide SCSI до 1994 года редко использовалась, поскольку эффективность их применения не была достаточно высокой. Однако широкое распространение дисковых массивов и дисковых накопителей со

скоростью вращения 7200 оборотов в минуту делают эту технологию весьма актуальной.
Следует отметить некоторую путаницу в терминологии. Часто стандартный 50-контактный разъем также называют разъемом SCSI-1, а более новый микроразъем - разъемом SCSI-2. Стандарт SCSI определяет только количество жил в кабеле, и вообще не определяет тип разъема.

9.3. 9.3. Устройства ввода/вывода

9.3.1. 9.3.1. Основные типы устройств ввода/вывода

Как правило, периферийные устройства компьютеров делятся на устройства ввода, устройства вывода и внешние запоминающие устройства (осуществляющие как ввод данных в машину, так и вывод данных из компьютера). Основной обобщающей характеристикой устройств ввода/вывода может служить скорость передачи данных (максимальная скорость, с которой данные могут передаваться между устройством ввода/вывода и основной памятью или процессором). На рис. 9.3. представлены основные устройства ввода/вывода, применяемые в современных компьютерах, а также указаны примерные скорости обмена данными, обеспечиваемые этими устройствами.

Тип устройства	Направление передачи данных	Скорость передачи данных (Кбайт/с)
Клавиатура	Ввод	0.01
Мышь	Ввод	0.02
Голосовой ввод	Ввод	0.02
Сканер	Ввод	200.0
Голосовой вывод	Вывод	0.06
Строчный принтер	Вывод	1.00
Лазерный принтер	Вывод	100.00
Графический дисплей	Вывод	30000.00
(ЦП → буфер кадра)	Вывод	200.0
Оптический диск	ЗУ	500.00
Магнитная лента	ЗУ	2000.00
Магнитный диск	ЗУ	2000.00

Рис. 9.3. Примеры устройств ввода/вывода

В рамках данного обзора мы рассмотрим наиболее быстрые из этих устройств: магнитные и магнитооптические диски, а также магнитные ленты.

9.3.2. 9.3.2. Магнитные и магнитооптические диски

В данном разделе мы кратко рассмотрим основную терминологию, применяемую при описании магнитных дисков и контроллеров, а затем приведем типовые характеристики нескольких современных дисковых подсистем.
Дисковый накопитель обычно состоит из набора пластин, представляющих собой металлические диски, покрытые магнитным материалом и соединенные между собой при помощи центрального шпинделя. Для записи данных используются обе поверхности пластины. В современных дисковых накопителях используется от 4 до 9

пластин. Шпиндель вращается с высокой постоянной скоростью (обычно 3600, 5400 или 7200 оборотов в минуту). Каждая пластина содержит набор концентрических записываемых дорожек. Обычно дорожки делятся на блоки данных объемом 512 байт, иногда называемые секторами. Количество блоков, записываемых на одну дорожку, зависит от физических размеров пластины и плотности записи.

Данные записываются или считываются с пластин с помощью головок записи/считывания, по одной на каждую поверхность. Линейный двигатель представляет собой электро-механическое устройство, которое позиционирует головку над заданной дорожкой. Обычно головки крепятся на кронштейнах, которые приводятся в движение каретками. Цилиндр - это набор дорожек, соответствующих одному положению каретки. Накопитель на магнитных дисках (НМД) представляет собой набор пластин, магнитных головок, кареток, линейных двигателей плюс воздухонепроницаемый корпус. Дисковым устройством называется НМД с относящимися к нему электронными схемами.

Производительность диска является функцией времени обслуживания, которое включает в себя три основных компонента: время доступа, время ожидания и время передачи данных. Время доступа - это время, необходимое для позиционирования головок на соответствующую дорожку, содержащую искомые данные. Оно является функцией затрат на начальные действия по ускорению головки диска (порядка 6 мс), а также функцией числа дорожек, которые необходимо пересечь на пути к искомой дорожке. Характерные средние времена поиска - время, необходимое для перемещения головки между двумя случайно выбранными дорожками, лежат в диапазоне 10-20 мс. Время перехода с дорожки на дорожку меньше 10 мс и обычно составляет 2 мс.

Вторым компонентом времени обслуживания является время ожидания. Чтобы искомый сектор повернулся до совмещения с положением головки, требуется некоторое время. После этого данные могут быть записаны или считаны. Для современных дисков время полного оборота лежит в диапазоне 8-16 мс, а среднее время ожидания составляет 4-8 мс.

Последним компонентом является время передачи данных, т.е. время, необходимое для физической передачи байтов. Время передачи данных является функцией от числа передаваемых байтов (размера блока), скорости вращения, плотности записи на дорожке и скорости электроники. Типичная скорость передачи равна 1-4 Мбайт/с.

В состав компьютеров часто входят специальные устройства, называемые дисковыми контроллерами. К каждому дисковому контроллеру может подключаться несколько дисковых накопителей. Между дисковым контроллером и основной памятью может быть целая иерархия контроллеров и магистралей данных, сложность которой определяется главным образом стоимостью компьютера. Поскольку время передачи часто составляет очень небольшую часть общего времени доступа к диску, контроллер в высокопроизводительной системе разъединяет магистрали данных от диска на время позиционирования так, что другие диски, подсоединенные к контроллеру, могут передавать свои данные в основную память. Поэтому время доступа к диску может увеличиваться на время, связанное с накладными расходами контроллера на организацию операции ввода/вывода.

Рассмотрим теперь основные составляющие времени доступа к диску в типичной подсистеме SCSI. Такая подсистема включает в себя четыре основных компонента: основной компьютер, главный адаптер SCSI, встроенный в дисковое устройство контроллер и собственно накопитель на магнитных дисках. Когда операционная система получает запрос от пользователя на выполнение операции ввода/вывода, она превращает этот запрос в набор команд SCSI. Запрашивающий процесс при этом блокируется и откладывается до завершения операции ввода/вывода (если только это был не запрос асинхронной передачи данных). Затем команды пересылаются по системе шин в главный адаптер SCSI, к которому подключен необходимый дисковый накопитель. После этого ответственность за выполнение взаимодействия с целевыми контроллерами и их устройствами ложится на главный адаптер.

Затем главный адаптер выбирает целевое устройство, устанавливая сигнал на линии управления шины SCSI (эта операция называется фазой выбора). Естественно, шина SCSI должна быть доступна для этой операции. Если целевое устройство возвращает ответ, то главный адаптер пересылает ему команду (это называется фазой команды). Если целевой контроллер может выполнить команду немедленно, то он пересылает в главный адаптер запрошенные данные или состояние. Команда может быть обслужена немедленно, только если это запрос состояния, или команда запрашивает данные, которые уже находятся в кэш-памяти целевого контроллера. Обычно же данные не доступны, и целевой контроллер выполняет разъединение, освобождая шину SCSI для других операций. Если выполняется операция записи, то за фазой команды на шине немедленно следует фаза данных, и данные помещаются в кэш-память целевого контроллера. Подтверждение записи обычно не происходит до тех пор, пока данные действительно не запишутся на поверхность диска.

После разъединения, целевой контроллер продолжает свою собственную работу. Если в нем не предусмотрены возможности буферизации команд (создание очереди команд), ему надо только выполнить одну команду. Однако, если создание очереди команд разрешено, то команда планируется в очереди работ целевого контроллера, при этом обрабатывается команда, обладающая наивысшим приоритетом в очереди. Когда запрос станет обладать наивысшим приоритетом, целевой контроллер должен вычислить физический адрес (или адреса), необходимый для обслуживания операции ввода/вывода. После этого становится доступным дисковый механизм: позиционируется каретка, подготавливается соответствующая головка записи/считывания и вычисляется момент появления данных под головкой. Наконец, данные физически считываются или записываются на дорожку. Считанные данные запоминаются в кэш-памяти целевого контроллера. Иногда целевой контроллер может выполнить считывание с просмотром вперед.

После завершения операции ввода/вывода целевой контроллер в случае свободы шины соединяется с главным адаптером, вслед за чем выполняется фаза данных (при передаче данных из целевого контроллера в главный адаптер) и фаза состояния для указания результата операции. Когда главный адаптер получает фазу состояния, он проверяет корректность завершения физической операции в целевом контроллере и соответствующим образом информирует операционную систему.

Одной из характеристик процесса ввода/вывода SCSI является большое количество шагов, которые обычно не видны пользователю. Обычно на шине SCSI происходит смена семи фаз (выбор, команда, разъединение, повторное соединение, данные, состояние, разъединение). Естественно каждая фаза выполняется за некоторое время, расходуемое на использование шины. Многие целевые контроллеры (особенно медленные устройства подобные магнитным лентам и компакт-дискам) потребляют значительную часть времени на реализацию фаз выбора, разъединения и повторного соединения.

Варианты применения высокопроизводительных подсистем ввода/вывода широко варьируются в зависимости от требований, которые к ним предъявляются. Они охватывают диапазон от обработки малого числа больших массивов данных, которые необходимо реализовать с минимальной задержкой (ввод/вывод суперкомпьютера), до большого числа простых заданий, которые оперируют с малыми объемами данных (обработка транзакций).

Запросы на ввод/вывод заданной рабочей нагрузки можно характеризовать в терминах трех метрик: производительность, время ожидания и пропускная способность. Производительность определяется числом запросов на обслуживание, получаемых в единицу времени. Время ожидания определяет время, необходимое на обслуживание индивидуального запроса. Пропускная способность определяет количество данных,

передаваемых между устройствами, требующими обслуживания, и устройствами, выполняющими обслуживание.
Ввод/вывод суперкомпьютера почти полностью определяется последовательным механизмом. Обычно данные передаются с диска в память большими блоками, а результаты записываются обратно на диск. В таких применениях требуется высокая пропускная способность и минимальное время ожидания, однако они характеризуются низкой производительностью. В отличие от этого обработка транзакций характеризуется огромным числом случайных обращений, относительно небольшими отрезками работы и требует умеренного времени ожидания при очень высокой производительности.
Так как системы обработки транзакций тратят большую часть времени обслуживания на поиск и ожидание, технологические успехи, приводящие к сокращению времени передачи, не будут оказывать особого влияния на производительность таких систем. С другой стороны, в научных применениях на поиск данных и на их передачу затрачивается одинаковое время, и поэтому производительность таких систем оказывается очень чувствительной к любым усовершенствованиям в технологии изготовления дисков. Как будет показано ниже, можно организовать матрицу дисков таким образом, что будет обеспечена высокая производительность ввода/вывода для широкого спектра рабочих нагрузок.
В последние годы плотность записи на жестких магнитных дисках увеличивается на 60% в год при ежеквартальном снижении стоимости хранения одного Мегабайта на 12%. По данным фирмы Dataquest такая тенденция сохранится и в ближайшие два года. Сейчас на рынке представлен широкий ассортимент дисковых накопителей емкостью до 9.1 Гбайт. При этом среднее время доступа у самых быстрых моделей достигает 8 мс. Например, жесткий диск компании Seagate Technology имеет емкость 4.1 Гбайт и среднее время доступа 8 мс при скорости вращения 7200 оборот/мин. Улучшаются также характеристики дисковых контроллеров на базе новых стандартов Fast SCSI-2 и Enhanced IDE. Предполагается увеличение скорости передачи данных до 13 Мбайт/с. Надежность жестких дисков также постоянно улучшается. Например, некоторые модели дисков компаний Conner Peripherals Inc., Micropolis Corp. и Hewlett-Packard имеют время наработки на отказ от 500 тысяч до 1 миллиона часов. На такие диски предоставляется 5-летняя гарантия.
Дальнейшее повышение надежности и коэффициента готовности дисковых подсистем достигается построением избыточных дисковых массивов RAID, о которых речь пойдет в подразделе 8.3.3.
Другим направлением развития систем хранения информации являются магнитооптические диски. Запись на магнитооптические диски (МО-диски) выполняется при взаимодействии лазера и магнитной головки. Луч лазера разогревает до точки Кюри (температуры потери материалом магнитных свойств) микроскопическую область записывающего слоя, которая при выходе из зоны действия лазера остывает, фиксируя магнитное поле, наведенное магнитной головкой. В результате данные, записанные на диск, не боятся сильных магнитных полей и колебаний температуры. Все функциональные свойства дисков сохраняются в диапазоне температур от -20 до +50 градусов Цельсия.
МО-диски уступают обычным жестким магнитным дискам лишь по времени доступа к данным. Предельное достигнутое МО-дисками время доступа составляет 19 мс. Магнитооптический принцип записи требует предварительного стирания данных перед записью, и соответственно, дополнительного оборота МО-диска. Однако завершенные недавно исследования в SONY и IBM показали, что это ограничение можно устранить, а плотность записи на МО-дисках можно увеличить в несколько раз. Во всех других отношениях МО-диски превосходят жесткие магнитные диски.
В магнитооптическом дисководе используются сменные диски, что обеспечивает практически неограниченную емкость. Стоимость хранения единицы данных на МО-

дисках в несколько раз меньше стоимости хранения того же объема данных на жестких магнитных дисках.

Сегодня на рынке МО-дисков предлагается более 150 моделей различных фирм. Одно из лидирующих положений на этом рынке занимает компания Pinnacle Micro Inc. Для примера, ее дисковод Sierra 1.3 Гбайт обеспечивает среднее время доступа 19 мс и среднее время наработки на отказ 80000 часов. Для серверов локальных сетей и рабочих станций компания Pinnacle Micro предлагает целый спектр многодисковых систем емкостью 20, 40, 120, 186 Гбайт и даже 4 Тбайт. Для систем высокой готовности Pinnacle Micro выпускает дисковый массив Array Optical Disk System, который обеспечивает эффективное время доступа к данным не более 11 мс при скорости передачи данных до 10 Мбайт/с.

9.3.3. 9.3.3. Дисковые массивы и уровни RAID

Одним из способов повышения производительности ввода/вывода является использование параллелизма путем объединения нескольких физических дисков в матрицу (группу) с организацией их работы аналогично одному логическому диску. К сожалению, надежность матрицы любых устройств падает при увеличении числа устройств. Полагая интенсивность отказов постоянной, т.е. при экспоненциальном законе распределения наработки на отказ, а также при условии, что отказы независимы, получим, что среднее время безотказной работы (mean time to failure - MTTF) матрицы дисков будет равно:

MTTF одного диска / Число дисков в матрице

Для достижения повышенного уровня отказоустойчивости приходится жертвовать пропускной способностью ввода/вывода или емкостью памяти. Необходимо использовать дополнительные диски, содержащие избыточную информацию, позволяющую восстановить исходные данные при отказе диска. Отсюда получают акроним для избыточных матриц недорогих дисков RAID (redundant array of inexpensive disks). Существует несколько способов объединения дисков RAID. Каждый уровень представляет свой компромисс между пропускной способностью ввода/вывода и емкостью диска, предназначенной для хранения избыточной информации.

Когда какой-либо диск отказывает, предполагается, что в течение короткого интервала времени он будет заменен и информация будет восстановлена на новом диске с использованием избыточной информации. Это время называется средним временем восстановления (mean time to repair - MTTR). Этот показатель можно уменьшить, если в систему входят дополнительные диски в качестве "горячего резерва": при отказе диска резервный диск подключается аппаратно-программными средствами. Периодически оператор вручную заменяет все отказавшие диски. Четыре основных этапа этого процесса состоят в следующем:

- • определение отказавшего диска,
- • устранение отказа без останова обработки;
- • восстановление потерянных данных на резервном диске;
- • периодическая замена отказавших дисков на новые.

RAID1: Зеркальные диски.

Зеркальные диски представляют традиционный способ повышения надежности магнитных дисков. Это наиболее дорогостоящий из рассматриваемых способов, так как все диски дублируются, и при каждой записи информация записывается также и на проверочный диск. Таким образом, приходится идти на некоторые жертвы в пропускной способности ввода/вывода и емкости памяти ради получения более высокой надежности. Зеркальные диски широко применяются многими фирмами. В частности компания Tandem Computers применяет зеркальные диски, а также дублирует контроллеры и магистрали ввода/вывода с целью повышения отказоустойчивости. Эта версия зеркальных дисков поддерживает параллельное считывание.

Контроллер HSC-70, используемый в VAX-кластерах компании DEC, выполнен по методу зеркальных дисков, называемому методом двойников. Содержимое отдельного диска распределяется между членами его группы двойников. Если группа состоит из двух двойников, мы получаем вариант зеркальных дисков. Заданный сектор может быть прочитан с любого из устройств группы двойников. После того как некоторый сектор записан, необходимо обновить информацию на всех дисках-двойниках. Контроллер имеет возможность предсказывать ожидаемые отказы некоторого диска и выделять горячий резерв для создания копии и сохранения ее на время работы механизма создания группы двойников. Затем отказавший диск может быть выключен.

Дублирование всех дисков может означать удвоение стоимости всей системы или, иначе, использование лишь 50% емкости диска для хранения данных. Повышение емкости, на которое приходится идти, составляет 100%. Такая низкая экономичность привела к появлению следующего уровня RAID.

RAID 2: матрица с поразрядным расслоением

Один из путей достижения надежности при снижении потерь емкости памяти может быть подсказан организацией основной памяти, в которой для исправления одиночных и обнаружения двойных ошибок используются избыточные контрольные разряды. Такое решение можно повторить путем поразрядного расслоения данных и записи их на диски группы, дополненной достаточным количеством контрольных дисков для обнаружения и исправления одиночных ошибок. Один диск контроля четности позволяет обнаружить одиночную ошибку, но для ее исправления требуется больше дисков.

Такая организация обеспечивает лишь один поток ввода/вывода для каждой группы независимо от ее размера. Группы большого размера приводят к снижению избыточной емкости, идущей на обеспечение отказоустойчивости, тогда как при организации меньшего числа групп наблюдается снижение операций ввода/вывода, которые могут выполняться матрицей параллельно.

При записи больших массивов данных системы уровня 2 имеют такую же производительность, что и системы уровня 1, хотя в них используется меньше контрольных дисков и, таким образом, по этому показателю они превосходят системы уровня 1. При передаче небольших порций данных производительность теряется, так как требуется записать либо считать группу целиком, независимо от конкретных потребностей. Таким образом, RAID уровня 2 предпочтительны для суперкомпьютеров, но не подходят для обработки транзакций. Компания Thinking Machine использовала RAID уровня 2 в ЭВМ Connection Machine при 32 дисках данных и 10 контрольных дисках, включая 3 диска горячего резерва.

RAID 3: аппаратное обнаружение ошибок и четность

Большинство контрольных дисков, используемых в RAID уровня 2, нужны для определения положения неисправного разряда. Эти диски становятся полностью избыточными, так как большинство контроллеров в состоянии определить, когда диск отказал при помощи специальных сигналов, поддерживаемых дисковым интерфейсом, либо при помощи дополнительного кодирования информации, записанной на диск и используемой для исправления случайных сбоев. По существу, если контроллер может определить положение ошибочного разряда, то для восстановления данных требуется лишь один бит четности. Уменьшение числа контрольных дисков до одного на группу снижает избыточность емкости до вполне разумных размеров. Часто количество дисков в группе равно 5 (4 диска данных плюс 1 контрольный). Подобные устройства выпускаются, например, фирмами Maxtor и Micropolis. Каждое из таких устройств воспринимается машиной как отдельный логический диск с учетверенной пропускной способностью, учетверенной емкостью и значительно более высокой надежностью.

RAID 4: внутригрупповой параллелизм

RAID уровня 4 повышает производительность передачи небольших объемов данных за счет параллелизма, давая возможность выполнять более одного обращения по вводу/выводу к группе в единицу времени. Логические блоки передачи в данном случае не распределяются между отдельными дисками, вместо этого каждый индивидуальный блок попадает на отдельный диск.

Достоинство поразрядного расслоения состоит в простоте вычисления кода Хэмминга, что необходимо для обнаружения и исправления ошибок в системах уровня 2. В RAID уровня 3 обнаружение ошибок диска с точностью до сектора осуществляется дисковым контроллером. Следовательно, если записывать отдельный блок передачи в отдельный сектор, то можно обнаружить ошибки отдельного считывания без доступа к дополнительным дискам. Главное отличие между системами уровня 3 и 4 состоит в том, что в последних расслоение выполняется на уровне сектора, а не на уровне битов или байтов.

В системах уровня 4 обновление контрольной информации реализовано достаточно просто. Для вычисления нового значения четности требуются лишь старый блок данных, старый блок четности и новый блок данных:

новая четность = (старые данные xor новые данные) xor старая четность

В системах уровня 4 для записи небольших массивов данных используются два диска, которые выполняют четыре выборки (чтение данных плюс четности, запись данных плюс четности). Производительность групповых операций записи и считывания остается прежней, но при небольших (на один диск) записях и считываниях производительность существенно улучшается. К сожалению, улучшение производительности оказывается недостаточной для того, чтобы этот метод мог занять место системы уровня 1.

RAID 5: четность вращения для распараллеливания записей

RAID уровня 4 позволяли добиться параллелизма при считывании отдельных дисков, но запись по-прежнему ограничена возможностью выполнения одной операции на группу, так как при каждой операции должны выполняться запись и чтение контрольного диска. Система уровня 5 улучшает возможности системы уровня 4 посредством распределения контрольной информации между всеми дисками группы.

Это небольшое изменение оказывает огромное влияние на производительность записи небольших массивов информации. Если операции записи могут быть спланированы так, чтобы обращаться за данными и соответствующими им блоками четности к разным дискам, появляется возможность параллельного выполнения N/2 записей, где N - число дисков в группе. Данная организация имеет одинаково высокую производительность при записи и при считывании как небольших, так и больших объемов информации, что делает ее наиболее привлекательной в случаях смешанных применений.

RAID 6: Двумерная четность для обеспечения большей надежности

Этот пункт можно рассмотреть в контексте соотношения отказоустойчивость/пропускная способность. RAID 5 предлагают, по существу, лишь одно измерение дисковой матрицы, вторым измерением которой являются секторы. Теперь рассмотрим объединение дисков в двумерный массив таким образом, чтобы секторы являлись третьим измерением. Мы можем иметь контроль четности по строкам, как в системах уровня 5, а также по столбцам, которые, в свою очередь. могут расслаиваться для обеспечения возможности параллельной записи. При такой организации можно преодолеть любые отказы двух дисков и многие отказы трех дисков. Однако при выполнении логической записи реально происходит шесть обращений к диску: за старыми данными, за четностью по строкам и по столбцам, а также для записи новых данных и новых значений четности. Для некоторых применений с очень высокими требованиями к отказоустойчивости такая избыточность может оказаться приемлемой, однако для традиционных суперкомпьютеров и для обработки транзакций данный метод не подойдет.

В общем случае, если доминируют короткие записи и считывания и стоимость емкости памяти не является определяющей, наилучшую производительность демонстрируют системы RAID уровня 1. Однако если стоимость емкости памяти существенна, либо если можно снизить вероятность появления коротких записей (например, при высоком коэффициенте отношения числа считываний к числу записей, при эффективной буферизации последовательностей считывания-модификации-записи, либо при приведении коротких записей к длинным с использованием стратегии кэширования файлов), RAID уровня 5 могут обеспечить очень высокую производительность, особенно в терминах отношения стоимость/производительность.

9.3.4. 9.3.4. Устройства архивирования информации

В качестве носителя для резервного копирования информации обычно используется магнитная лента. Резервное копирование предполагает использование различных стратегий и различных конфигураций оборудования в зависимости от требований пользователя. При планировании и создании системы этим вопросам приходится уделять большое внимание, так как обычно требования к системе резервного копирования выходят далеко за рамки простого обеспечения емкости носителя, превышающей емкость дисковой памяти системы, или выбора скорости операций копирования на магнитную ленту. Среди этих вопросов следует выделить, например, такие как определение количества клиентов, копирование данных которых должно осуществляться одновременно; цикличность операций копирования, т.е. по каким дням и в какие часы такое копирование должно осуществляться, а также уровень копирования (полное, частичное или смешанное); определение устройств на которых должно выполняться резервное копирование и т.д.
В настоящее время в большинстве систем накопители на магнитных лентах (НМЛ) обычно подсоединяются к компьютеру с помощью шины SCSI. Очень часто к этой же шине подсоединяются и дисковые накопители. К сожалению, высокий коэффициент использования шины SCSI практически всеми применяемыми в настоящее время типами НМЛ становится критическим фактором при организации резервного копирования и восстановления информации, особенно в больших серверах с высокой степенью готовности. В таблице 9.1 приведены типичные параметры НМЛ. Очевидно такая высокая загрузка шины SCSI (до 20 - 65 % пропускной способности шины) при работе НМЛ накладывает определенные ограничения как на конфигурацию и типы применяемых НМЛ, так и на организацию самого резервного копирования.

Таблица 9.1

Тип НДЛ	Емкость	Скорость передачи данных	Скорость пересылки по шине	Коэффициент использования шины SCSI
4 мм	5 Гб	920 Кб/с	5 Мб/с (синх.)	25 %
8 мм	2.3 Гб	220 Кб/с	1.2 Мб/с (асинх.)	25 %
8 мм	5 Гб	500 Кб/с	3 Мб/с (асинх.)	20 %
1/2” 9 дор.	120 Мб	780 Кб/с	1.2 Мб/с (асинх.)	65-75 %
1/4” QIC	150 Мб	200 Кб/с	1.0 Мб/с (асинх.)	28 %

Наиболее популярным в настоящее время являются НМЛ с 8 и 4 мм цифровой аудио-лентой (DAT), использующие технологию спирального сканирования. В отличие от традиционных НМЛ со стационарными головками и ограниченным числом дорожек, эти устройства осуществляют чтение и запись данных на медленно двигающуюся магнитную ленту с помощью головок, размещаемых на быстро вращающемся барабане. При этом дорожки пересекают ленту с края на край и расположены под небольшим углом к направлению, перпендикулярному направлению движения ленты. Иногда эту технологию называют "поперечной записью". На сегодняшний день подобные устройства дают наивысшую поверхностную плотность записи. Например, накопитель EXB-8200 компании Exabyte Corp. позволяет записывать около 35 мегабит на квадратный дюйм 8 мм ленты, а накопитель EXB-8500 - около 75 мегабит на квадратный дюйм. Устройства DAT записывают данные на 4 мм ленту с плотностью 114 мегабит на дюйм, что близко к теоретическому пределу плотности записи. Дальнейшее ее увеличение требует смены типа носителя или использования технологии компрессии (сжатия) данных.

На сегодняшний день продолжают использоваться и старые типы катушечных НМЛ, которые используют стандартную магнитную ленту шириной 0.5 дюйма. Они главным образом применяются для обмена информацией со старыми ЭВМ и поддерживают плотность записи 6250, 1600 и 800 бит на дюйм.

Наиболее популярными в течение многих лет были 150-250 Мб картриджи QIC с лентой шириной 1/4 дюйма. В настоящее время существует 10 производственных стандартов для картриджей конструктива 5.25" и 9 стандартов мини-картриджей конструктива 3.5". В мае 1994 года появился новый формат для записи 2 Гбайт (без сжатия) на микрокартридже QIC-153 с барий-ферритовой лентой длиной 400 футов. QIC-картриджи вмещают до 1200 футов магнитной ленты, при этом данные записываются на дорожках, расположенных вдоль ленты. Число дорожек может достигать 48. В зависимости от формата (QIC-40, QIC-80, QIC-3GB(M) и т.д.) мини-картриджи имеют емкость (без сжатия) от 40 Мбайт до 3 и более Гбайт. Картриджи наибольшей емкости позволяют записать до 13 Гбайт данных. В настоящее время наблюдается рост числа накопителей QIC с картриджами емкостью до 5 Гбайт и форматом записи 5GB(M). В 1995 году ожидается появление накопителей QIC формата 25 Мбайт с постоянной скоростью передачи 2.4 Мбайт/с. Такие системы составят серьезную конкуренцию 8 мм накопителям типа Exabyte, которые сейчас доминируют на рынке систем хранения большой емкости.

Одним из сравнительно новых направлений в области резервного копирования является появление устройств ленточных массивов (аналогичных дисковым массивам), используемых главным образом в системах высокой готовности. Примером такого устройства может служить CLARiiON Series 4000 tape array компании Data General. Оно может иметь в своем составе до пяти 4 мм DAT накопителей общей емкостью до 25 Гбайт. Устройство относится к разряду открытых систем и совместимо со всеми UNIX-платформами компаний IBM, Sun, Hewlett-Packard, Unisys и ICL.

10. 10. Многопроцессорные системы

6.5. 10.1. Классификация систем параллельной обработки данных

На протяжении всей истории развития вычислительной техники делались попытки найти какую-то общую классификацию, под которую подпадали бы все возможные направления развития компьютерных архитектур. Ни одна из таких классификаций не могла охватить все разнообразие разрабатываемых архитектурных решений и не выдерживала испытания временем. Тем не менее в научный оборот попали и широко используются ряд терминов, которые полезно знать не только разработчикам, но и пользователям компьютеров.

Любая вычислительная система (будь то супер-ЭВМ или персональный компьютер) достигает своей наивысшей производительности благодаря использованию высокоскоростных элементов и параллельному выполнению большого числа операций. Именно возможность параллельной работы различных устройств системы (работы с перекрытием) является основой ускорения основных операций.

Параллельные ЭВМ часто подразделяются по классификации Флинна на машины типа SIMD (Single Instruction Multiple Data - с одним потоком команд при множественном потоке данных) и MIMD (Multiple Instruction Multiple Data - с множественным потоком команд при множественном потоке данных). Как и любая другая, приведенная выше классификация несовершенна: существуют машины прямо в нее не попадающие, имеются также важные признаки, которые в этой классификации не учтены. В частности, к машинам типа SIMD часто относят векторные процессоры, хотя их высокая производительность зависит от другой формы параллелизма - конвейерной организации машины. Многопроцессорные векторные системы, типа Cray Y-MP, состоят из нескольких векторных процессоров и поэтому могут быть названы MSIMD (Multiple SIMD).

Классификация Флинна не делает различия по другим важным для вычислительных моделей характеристикам, например, по уровню "зернистости" параллельных вычислений и методам синхронизации.

Можно выделить четыре основных типа архитектуры систем параллельной обработки:

1) Конвейерная и векторная обработка.

Основу конвейерной обработки составляет раздельное выполнение некоторой операции в несколько этапов (за несколько ступеней) с передачей данных одного этапа следующему. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько операций. Конвейеризация эффективна только тогда, когда загрузка конвейера близка к полной, а скорость подачи новых операндов соответствует максимальной производительности конвейера. Если происходит задержка, то параллельно будет выполняться меньше операций и суммарная производительность снизится. Векторные операции обеспечивают идеальную возможность полной загрузки вычислительного конвейера.

При выполнении векторной команды одна и та же операция применяется ко всем элементам вектора (или чаще всего к соответствующим элементам пары векторов). Для настройки конвейера на выполнение конкретной операции может потребоваться некоторое установочное время, однако затем операнды могут поступать в конвейер с максимальной скоростью, допускаемой возможностями памяти. При этом не возникает пауз ни в связи с выборкой новой команды, ни в связи с определением ветви вычислений при условном переходе. Таким образом, главный принцип вычислений на векторной машине состоит в выполнении некоторой элементарной операции или комбинации из нескольких элементарных операций, которые должны повторно применяться к некоторому блоку данных. Таким операциям в исходной программе соответствуют небольшие компактные циклы.

2) Машины типа SIMD. Машины типа SIMD состоят из большого числа идентичных процессорных элементов, имеющих собственную память. Все процессорные элементы в такой машине выполняют одну и ту же программу. Очевидно, что такая машина, составленная из большого числа процессоров, может обеспечить очень высокую производительность только на тех задачах, при решении которых все процессоры могут делать одну и ту же работу. Модель вычислений для машины SIMD очень похожа на модель вычислений для векторного процессора: одиночная операция выполняется над большим блоком данных.

В отличие от ограниченного конвейерного функционирования векторного процессора, матричный процессор (синоним для большинства SIMD-машин) может быть значительно более гибким. Обрабатывающие элементы таких процессоров - это универсальные программируемые ЭВМ, так что задача, решаемая параллельно, может быть достаточно сложной и содержать ветвления. Обычное проявление этой вычислительной модели в исходной программе примерно такое же, как и в случае векторных операций: циклы на элементах массива, в которых значения, вырабатываемые на одной итерации цикла, не используются на другой итерации цикла.

Модели вычислений на векторных и матричных ЭВМ настолько схожи, что эти ЭВМ часто обсуждаются как эквивалентные.

3) Машины типа MIMD. Термин "мультипроцессор" покрывает большинство машин типа MIMD и (подобно тому, как термин "матричный процессор" применяется к машинам типа SIMD) часто используется в качестве синонима для машин типа MIMD. В мультипроцессорной системе каждый процессорный элемент (ПЭ) выполняет свою программу достаточно независимо от других процессорных элементов. Процессорные элементы, конечно, должны как-то связываться друг с другом, что делает необходимым более подробную классификацию машин типа MIMD. В мультипроцессорах с общей памятью (сильносвязанных мультипроцессорах) имеется память данных и команд, доступная всем ПЭ. С общей памятью ПЭ связываются с помощью общей шины или сети обмена. В противоположность этому варианту в слабосвязанных многопроцессорных системах (машинах с локальной памятью) вся память делится между процессорными элементами и каждый блок памяти доступен только связанному с ним процессору. Сеть обмена связывает процессорные элементы друг с другом.

Базовой моделью вычислений на MIMD-мультипроцессоре является совокупность независимых процессов, эпизодически обращающихся к разделяемым данным. Существует большое количество вариантов этой модели. На одном конце спектра - модель распределенных вычислений, в которой программа делится на довольно большое число параллельных задач, состоящих из множества подпрограмм. На другом конце спектра - модель потоковых вычислений, в которых каждая операция в программе может рассматриваться как отдельный процесс. Такая операция ждет своих входных данных (операндов), которые должны быть переданы ей другими процессами. По их получении операция выполняется, и полученное значение передается тем процессам, которые в нем нуждаются. В потоковых моделях вычислений с большим и средним уровнем гранулярности, процессы содержат большое число операций и выполняются в потоковой манере.

4) Многопроцессорные машины с SIMD-процессорами.

Многие современные супер-ЭВМ представляют собой многопроцессорные системы, в которых в качестве процессоров используются векторные процессоры или процессоры типа SIMD. Такие машины относятся к машинам класса MSIMD.

Языки программирования и соответствующие компиляторы для машин типа MSIMD обычно обеспечивают языковые конструкции, которые позволяют программисту описывать "крупнозернистый" параллелизм. В пределах каждой задачи компилятор автоматически векторизует подходящие циклы. Машины типа MSIMD, как можно себе представить, дают возможность использовать лучший из этих двух принципов декомпозиции: векторные операции ("мелкозернистый" параллелизм) для тех частей программы, которые подходят для этого, и гибкие возможности MIMD-архитектуры для других частей программы.

Многопроцессорные системы за годы развития вычислительной техники претерпели ряд этапов своего развития. Исторически первой стала осваиваться технология SIMD. Однако в настоящее время наметился устойчивый интерес к архитектурам MIMD. Этот интерес главным образом определяется двумя факторами:

1. 1. Архитектура MIMD дает большую гибкость: при наличии адекватной поддержки со стороны аппаратных средств и программного обеспечения MIMD может работать как однопользовательская система, обеспечивая высокопроизводительную обработку данных для одной прикладной задачи, как многопрограммная машина, выполняющая множество задач параллельно, и как некоторая комбинация этих возможностей.
2. 2. Архитектура MIMD может использовать все преимущества современной микропроцессорной технологии на основе строгого учета соотношения стоимость/производительность. В действительности практически все современные многопроцессорные системы строятся на тех же микропроцессорах, которые можно найти в персональных компьютерах, рабочих станциях и небольших однопроцессорных серверах.

Одной из отличительных особенностей многопроцессорной вычислительной системы является сеть обмена, с помощью которой процессоры соединяются друг с другом или с памятью. Модель обмена настолько важна для многопроцессорной системы, что многие характеристики производительности и другие оценки выражаются отношением времени обработки к времени обмена, соответствующим решаемым задачам. Существуют две основные модели межпроцессорного обмена: одна основана на передаче сообщений, другая - на использовании общей памяти. В многопроцессорной системе с общей памятью один процессор осуществляет запись в конкретную ячейку, а другой процессор производит считывание из этой ячейки памяти. Чтобы обеспечить согласованность данных и синхронизацию процессов, обмен часто реализуется по принципу взаимно исключающего доступа к общей памяти методом "почтового ящика".

В архитектурах с локальной памятью непосредственное разделение памяти невозможно. Вместо этого процессоры получают доступ к совместно используемым данным посредством передачи сообщений по сети обмена. Эффективность схемы коммуникаций зависит от протоколов обмена, основных сетей обмена и пропускной способности памяти и каналов обмена.

Часто, и притом не обосновано, в машинах с общей памятью и векторных машинах затраты на обмен не учитываются, так как проблемы обмена в значительной степени скрыты от программиста. Однако накладные расходы на обмен в этих машинах имеются и определяются конфликтами шин, памяти и процессоров. Чем больше процессоров добавляется в систему, тем больше процессов соперничают при использовании одних и тех же данных и шины, что приводит к состоянию насыщения. Модель системы с общей памятью очень удобна для программирования и иногда рассматривается как высокоуровневое средство оценки влияния обмена на работу системы, даже если основная система в действительности реализована с применением локальной памяти и принципа передачи сообщений.

В сетях с коммутацией каналов и в сетях с коммутацией пакетов по мере возрастания требований к обмену следует учитывать возможность перегрузки сети. Здесь межпроцессорный обмен связывает сетевые ресурсы: каналы, процессоры, буферы сообщений. Объем передаваемой информации может быть сокращен за счет тщательной функциональной декомпозиции задачи и тщательного диспетчирования выполняемых функций.

Таким образом, существующие MIMD-машины распадаются на два основных класса в зависимости от количества объединяемых процессоров, которое определяет и способ организации памяти, и методику их межсоединений.

К первой группе относятся машины с общей (разделяемой) основной памятью, объединяющие до нескольких десятков (обычно менее 32) процессоров. Сравнительно небольшое количество процессоров в таких машинах позволяет иметь одну централизованную общую память и объединить процессоры и память с помощью одной шины. При наличии у процессоров кэш-памяти достаточного объема высокопроизводительная шина и общая память могут удовлетворить обращения к памяти, поступающие от нескольких процессоров. Поскольку имеется единственная память с одним и тем же временем доступа, эти машины иногда называются UMA (Uniform Memory Access). Такой способ организации со сравнительно небольшой разделяемой памятью в настоящее время является наиболее популярным. Структура подобной системы представлена на рис. 10.1.

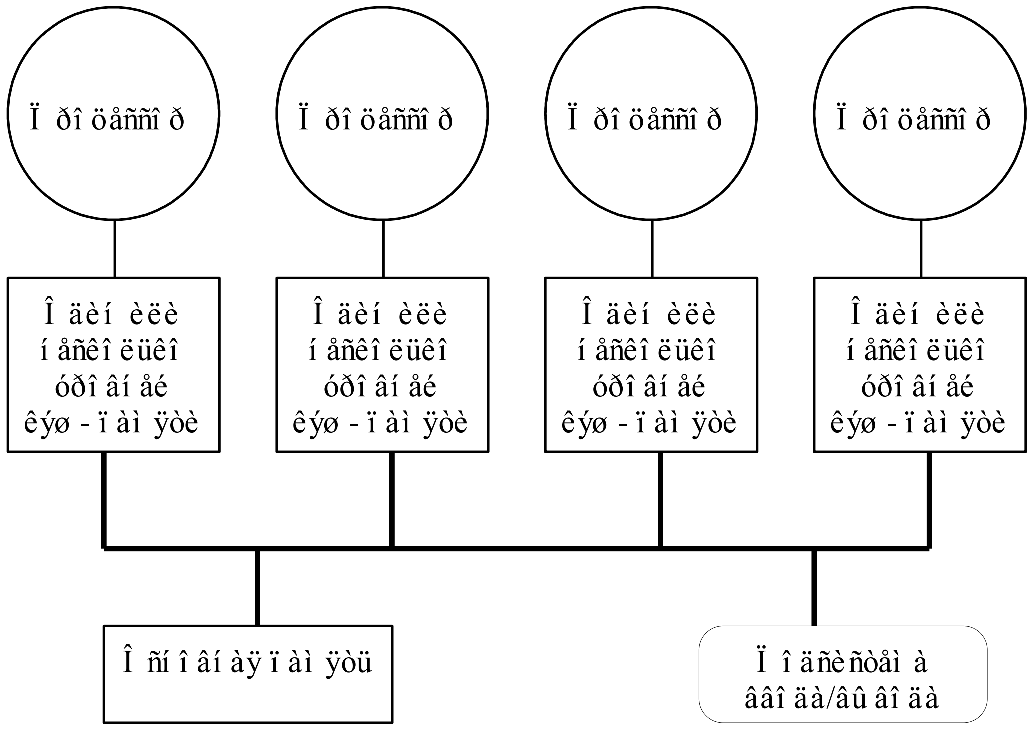


Рис. 10.1. Типовая архитектура мультипроцессорной системы с общей памятью.

Вторую группу машин составляют крупномасштабные системы с распределенной памятью. Для того чтобы поддерживать большое количество процессоров приходится распределять основную память между ними, в противном случае полосы пропускания памяти просто может не хватить для удовлетворения запросов, поступающих от очень большого числа процессоров. Естественно при таком подходе также требуется реализовать связь процессоров между собой. На рис. 10.2 показана структура такой системы.

С ростом числа процессоров просто невозможно обойти необходимость реализации модели распределенной памяти с высокоскоростной сетью для связи процессоров. С быстрым ростом производительности процессоров и связанным с этим ужесточением требования увеличения полосы пропускания памяти, масштаб систем (т.е. число процессоров в системе), для которых требуется организация распределенной памяти, уменьшается, также как и уменьшается число процессоров, которые удается поддерживать на одной разделяемой шине и общей памяти.

Распределение памяти между отдельными узлами системы имеет два главных преимущества. Во-первых, это эффективный с точки зрения стоимости способ увеличения полосы пропускания памяти, поскольку большинство обращений могут выполняться параллельно к локальной памяти в каждом узле. Во-вторых, это уменьшает задержку обращения (время доступа) к локальной памяти. Эти два преимущества еще больше сокращают количество процессоров, для которых архитектура с распределенной памятью имеет смысл.

Обычно устройства ввода/вывода, также как и память, распределяются по узлам и в действительности узлы могут состоять из небольшого числа (2-8) процессоров, соединенных между собой другим способом. Хотя такая кластеризация нескольких процессоров с памятью и сетевой интерфейс могут быть достаточно полезными с точки зрения эффективности в стоимостном выражении, это не очень существенно для понимания того, как такая машина работает, поэтому мы пока остановимся на системах с одним процессором на узел. Основная разница в архитектуре, которую следует выделить в машинах с распределенной памятью, заключается в том, как осуществляется связь и какова логическая модель памяти.

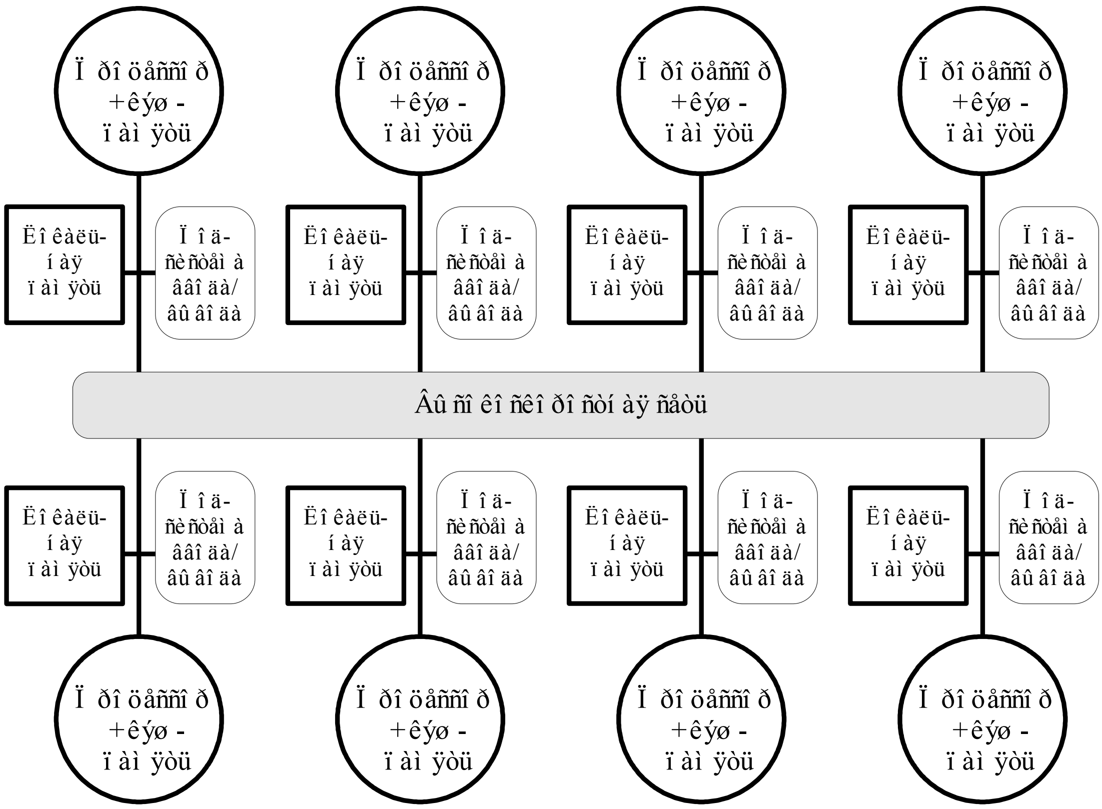


Рис. 10.2. Типовая архитектура машины с распределенной памятью.

Модели связи и архитектуры памяти

Как уже было отмечено, любая крупномасштабная многопроцессорная система должна использовать множество устройств памяти, которые физически распределяются вместе с процессорами. Имеется две альтернативных организации адресации этих устройств памяти и связанных с этим два альтернативных метода для передачи данных между процессорами. Физически отдельные устройства памяти могут адресоваться как логически единое адресное пространство, что означает, что любой процессор может выполнять обращения к любым ячейкам памяти, предполагая, что он имеет соответствующие права доступа. Такие машины называются машинами с распределенной разделяемой (общей) памятью (DSM - distributed shared memory), масштабируемые архитектуры с разделяемой памятью, а иногда NUMA's - Non-Uniform Memory Access, поскольку время доступа зависит от расположения ячейки в памяти.

В альтернативном случае, адресное пространство состоит из отдельных адресных пространств, которые логически не связаны и доступ к которым не может быть осуществлен аппаратно другим процессором. В таком примере каждый модуль процессор-память представляет собой отдельный компьютер, поэтому такие системы называются многомашинными (multicomputers).

С каждой из этих организаций адресного пространства связан свой механизм обмена. Для машины с единым адресным пространством это адресное пространство может быть использовано для обмена данными посредством операций загрузки и записи. Поэтому эти машины и получили название машин с разделяемой (общей) памятью. Для машин с множеством адресных пространств обмен данными должен использовать другой механизм: передачу сообщений между процессорами; поэтому эти машины часто называют машинами с передачей сообщений.

Каждый из этих механизмов обмена имеет свои преимущества. Для обмена в общей памяти это включает:

- • Совместимость с хорошо понятными используемыми как в однопроцессорных, так и маломасштабных многопроцессорных системах, механизмами, которые используют для обмена общую память.
- • Простота программирования, когда модели обмена между процессорами сложные или динамически меняются во время выполнения. Подобные преимущества упрощают конструирование компилятора.
- • Более низкая задержка обмена и лучшее использование полосы пропускания при обмене малыми порциями данных.
- • Возможность использования аппаратно управляемого кэширования для снижения частоты удаленного обмена, допускающая кэширование всех данных как разделяемых, так и неразделяемых.

Основные преимущества обмена с помощью передачи сообщений являются:

- • Аппаратура может быть более простой, особенно по сравнению с моделью разделяемой памяти, которая поддерживает масштабируемую когерентность кэш-памяти.
- • Модели обмена понятны, принуждают программистов (или компиляторы) уделять внимание обмену, который обычно имеет высокую, связанную с ним стоимость.

Конечно, требуемая модель обмена может быть надстроена над аппаратной моделью, которая использует любой из этих механизмов. Поддержка передачи сообщений над разделяемой памятью, естественно, намного проще, если предположить, что машины имеют адекватные полосы пропускания. Основные трудности возникают при работе с сообщениями, которые могут быть неправильно выровнены и сообщениями произвольной длины в системе памяти, которая обычно ориентирована на передачу выровненных блоков данных, организованных как блоки кэш-памяти. Эти трудности можно преодолеть либо с небольшими потерями производительности программным способом, либо существенно без потерь при использовании небольшой аппаратной поддержки.

Построение механизмов реализации разделяемой памяти над механизмом передачи сообщений намного сложнее. Без предполагаемой поддержки со стороны аппаратуры все обращения к разделяемой памяти потребуют привлечения операционной системы как для обеспечения преобразования адресов и защиты памяти, так и для преобразования обращений к памяти в посылку и прием сообщений. Поскольку операции загрузки и записи обычно работают с небольшим объемом данных, то большие накладные расходы по поддержанию такого обмена делают невозможной чисто программную реализацию.

При оценке любого механизма обмена критичными являются три характеристики производительности:

1. 1. *Полоса пропускания:* в идеале полоса пропускания механизма обмена будет ограничена полосами пропускания процессора, памяти и системы межсоединений, а не какими-либо аспектами механизма обмена. Связанные с механизмом обмена накладные расходы (например, длина межпроцессорной связи) прямо воздействуют на полосу пропускания.
2. 2. *Задержка:* в идеале задержка должна быть настолько мала, насколько это возможно. Для ее определения критичны накладные расходы аппаратуры и программного обеспечения, связанные с инициированием и завершением обмена.
3. 3. *Упрятывание задержки:* насколько хорошо механизм скрывает задержку путем перекрытия обмена с вычислениями или с другими обменами.

Каждый из этих параметров производительности воздействует на характеристики обмена. В частности, задержка и полоса пропускания могут меняться в зависимости от размера элемента данных. В общем случае, механизм, который одинаково хорошо работает как с небольшими, так и с большими объемами данных будет более гибким и эффективным.

Таким образом, отличия разных машин с распределенной памятью определяются моделью памяти и механизмом обмена. Исторически машины с распределенной памятью первоначально были построены с использованием механизма передачи сообщений, поскольку это было очевидно проще, и многие разработчики и исследователи не верили, что единое адресное пространство можно построить и в машинах с распределенной памятью. С недавнего времени модели обмена с общей памятью действительно начали поддерживаться практически в каждой разработанной машине (характерным примером могут служить системы с симметричной мультипроцессорной обработкой). Хотя машины с централизованной общей памятью, построенные на базе общей шины все еще доминируют в терминах размера компьютерного рынка, долговременные технические тенденции направлены на использование преимуществ распределенной памяти даже в машинах умеренного размера. Как мы увидим, возможно наиболее важным вопросом, который встает при создании машин с распределенной памятью, является вопрос о кэшировании и когерентности кэш-памяти.

6.6. **10.2. Многопроцессорные системы с общей памятью**

Требования, предъявляемые современными процессорами к полосе пропускания памяти можно существенно сократить путем применения больших многоуровневых кэшей. Тогда, если эти требования снижаются, то несколько процессоров смогут разделять доступ к одной и той же памяти. Начиная с 1980 года, эта идея, подкрепленная широким распространением микропроцессоров, стимулировала многих разработчиков на создание небольших мультипроцессоров, в которых несколько процессоров разделяют одну физическую память, соединенную с ними с помощью разделяемой шины. Из-за малого размера процессоров и заметного сокращения требуемой полосы пропускания шины, достигнутого за счет возможности реализации достаточно большой кэш-памяти, такие машины стали исключительно эффективными по стоимости. В первых разработках подобного рода машин удавалось разместить весь процессор и кэш на одной плате, которая затем вставлялась в заднюю панель, с помощью которой реализовывалась шинная архитектура. Современные конструкции позволяют разместить до четырех процессоров на одной плате. На рис. 10.1 показана схема именно такой машины.

В такой машине кэши могут содержать как разделяемые, так и частные данные. Частные данные - это данные, которые используются одним процессором, в то время как разделяемые данные используются многими процессорами, по существу обеспечивая обмен между ними. Когда кэшируется элемент частных данных, их значение переносится в кэш для сокращения среднего времени доступа, а также требуемой полосы пропускания. Поскольку никакой другой процессор не использует эти данные, этот процесс идентичен процессу для однопроцессорной машины с кэш-памятью. Если кэшируются разделяемые данные, то разделяемое значение реплицируется и может содержаться в нескольких кэшах. Кроме сокращения задержки доступа и требуемой полосы пропускания такая репликация данных способствует также общему сокращению количества обменов. Однако кэширование разделяемых данных вызывает новую проблему: когерентность кэш-памяти.

Мультипроцессорная когерентность кэш-памяти

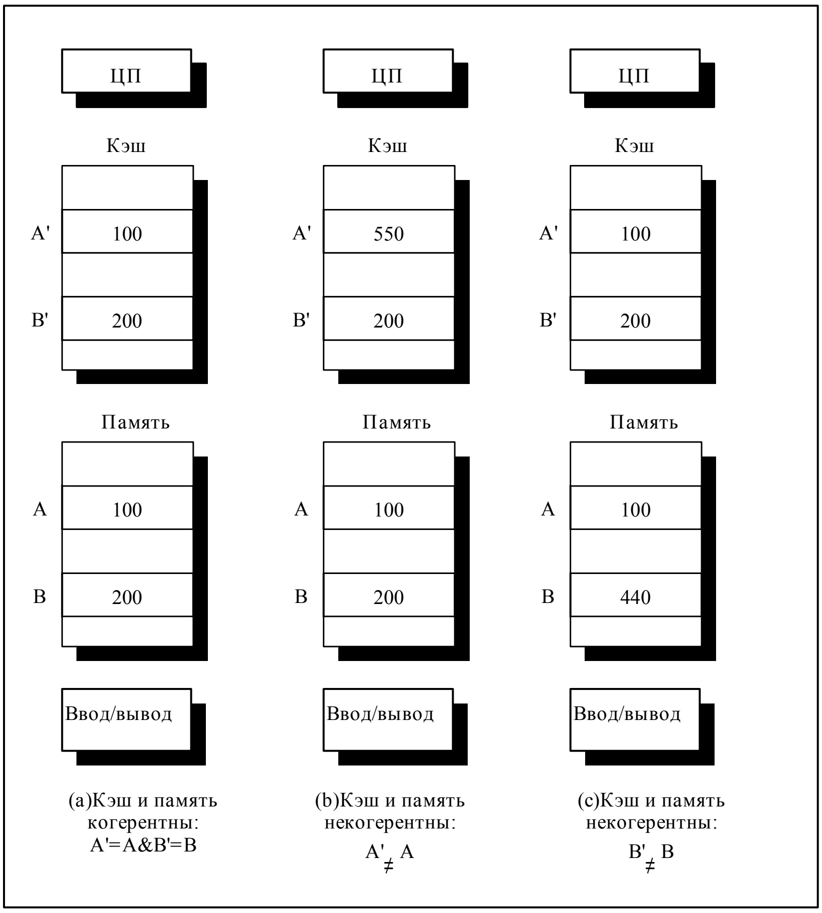
Проблема, о которой идет речь, возникает из-за того, что значение элемента данных в памяти, хранящееся в двух разных процессорах, доступно этим процессорам только через их индивидуальные кэши. На рис. 10.3 показан простой пример, иллюстрирующий эту проблему.

Проблема когерентности памяти для мультипроцессоров и устройств ввода/вывода имеет много аспектов. Обычно в малых мультипроцессорах используется аппаратный механизм, называемый протоколом, позволяющий решить эту проблему. Такие протоколы называются протоколами когерентности кэш-памяти. Существуют два класса таких протоколов:

1. 1. Протоколы на основе справочника (directory based). Информация о состоянии блока физической памяти содержится только в одном месте, называемом справочником (физически справочник может быть распределен по узлам системы). Этот подход будет рассмотрен в разд. 10.3.
2. 2. Протоколы наблюдения (snooping). Каждый кэш, который содержит копию данных некоторого блока физической памяти, имеет также соответствующую копию служебной информации о его состоянии. Централизованная система записей отсутствует. Обычно кэши расположены на общей (разделяемой) шине и контроллеры всех кэшей наблюдают за шиной (просматривают ее) для определения того, не содержат ли они копию соответствующего блока.

В мультипроцессорных системах, использующих микропроцессоры с кэш-памятью, подсоединенные к централизованной общей памяти, протоколы наблюдения приобрели популярность, поскольку для опроса состояния кэшей они могут использовать заранее существующее физическое соединение - шину памяти.

Неформально, проблема когерентности памяти состоит в необходимости гарантировать, что любое считывание элемента данных возвращает последнее по времени записанное в него значение. Это определение не совсем корректно, поскольку невозможно требовать, чтобы операция считывания мгновенно видела значение, записанное в этот элемент данных некоторым другим процессором. Если, например, операция записи на одном процессоре предшествует операции чтения той же ячейки на другом процессоре в пределах очень короткого интервала времени, то невозможно гарантировать, что чтение вернет записанное значение данных, поскольку в этот момент времени записываемые данные могут даже не покинуть процессор. Вопрос о том, когда точно записываемое значение должно быть доступно процессору, выполняющему чтение, определяется выбранной моделью согласованного (непротиворечивого) состояния памяти и связан с реализацией синхронизации параллельных вычислений. Поэтому с целью упрощения предположим, что мы требуем только, чтобы записанное операцией записи значение было доступно операции чтения, возникшей немного позже записи и что операции записи данного процессора всегда видны в порядке их выполнения.



A' и B' - кэшированные копии элементов A и B в основной памяти

a) Когерентное состояние кэша и основной памяти.

b) Предполагается использование кэш-памяти с отложенным обратным копированием, когда ЦП записывает значение 550 в ячейку A. В результате A' содержит новое значение, а в основной памяти осталось старое значение 100. При попытке вывода A из памяти будет получено старое значение.

c) Подсистема ввода/вывода вводит в ячейку памяти B новое значение 440, а в кэш-памяти осталось старое значение B.

Рис. 10.3. Иллюстрация проблемы когерентности кэш-памяти

С этим простым определением согласованного состояния памяти мы можем гарантировать когерентность путем обеспечения двух свойств:

1. 1. Операция чтения ячейки памяти одним процессором, которая следует за операцией записи в ту же ячейку памяти другим процессором, получит записанное значение, если операции чтения и записи достаточно отделены друг от друга по времени.
2. 2. Операции записи в одну и ту же ячейку памяти выполняются строго последовательно (иногда говорят, что они сериализованы). Это означает, что две подряд идущие операции записи в одну и ту же ячейку памяти будут наблюдаться другими процессорами именно в том порядке, в котором они появляются в программе процессора, выполняющего эти операции записи.

Первое свойство очевидно связано с определением когерентного (согласованного) состояния памяти: если бы процессор всегда бы считывал только старое значение данных, мы сказали бы, что память некогерентна.

Необходимость строго последовательного выполнения операций записи является более тонким, но также очень важным свойством. Представим себе, что строго последовательное выполнение операций записи не соблюдается. Тогда процессор P1 может записать данные в ячейку, а затем в эту ячейку выполнит запись процессор P2. Строго последовательное выполнение операций записи гарантирует два важных следствия для этой последовательности операций записи. Во-первых, оно гарантирует, что каждый процессор в машине в некоторый момент времени будет наблюдать запись, выполняемую процессором P2. Если последовательность операций записи не соблюдается, то может возникнуть ситуация, когда какой-нибудь процессор будет наблюдать сначала операцию записи процессора P2, а затем операцию записи процессора P1, и будет хранить это записанное P1 значение неограниченно долго. Более тонкая проблема возникает с поддержанием разумной модели порядка выполнения программ и когерентности памяти для пользователя: представьте, что третий процессор постоянно читает ту же самую ячейку памяти, в которую записывают процессоры P1 и P2; он должен наблюдать сначала значение, записанное P1, а затем значение, записанное P2. Возможно он никогда не сможет увидеть значения, записанного P1, поскольку запись от P2 возникла раньше чтения. Если он даже видит значение, записанное P1, он должен видеть значение, записанное P2, при последующем чтении. Подобным образом любой другой процессор, который может наблюдать за значениями, записываемыми как P1, так и P2, должен наблюдать идентичное поведение. Простейший способ добиться таких свойств заключается в строгом соблюдении порядка операций записи, чтобы все записи в одну и ту же ячейку могли наблюдаться в том же самом порядке. Это свойство называется последовательным выполнением (сериализацией) операций записи (write serialization). Вопрос о том, когда процессор должен увидеть значение, записанное другим процессором, достаточно сложен и имеет заметное воздействие на производительность, особенно в больших машинах.

Альтернативные протоколы

Имеются две методики поддержания описанной выше когерентности. Один из методов заключается в том, чтобы гарантировать, что процессор должен получить исключительные права доступа к элементу данных перед выполнением записи в этот элемент данных. Этот тип протоколов называется протоколом *записи с аннулированием (write ivalidate protocol),* поскольку при выполнении записи он аннулирует другие копии. Это наиболее часто используемый протокол как в схемах на основе справочников, так и в схемах наблюдения. Исключительное право доступа гарантирует, что во время выполнения записи не существует никаких других копий элемента данных, в которые можно писать или из которых можно читать: все другие кэшированные копии элемента данных аннулированы. Чтобы увидеть, как такой протокол обеспечивает когерентность, рассмотрим операцию записи, вслед за которой следует операция чтения другим процессором. Поскольку запись требует исключительного права доступа, любая копия, поддерживаемая читающим процессором, должна быть аннулирована (в соответствии с названием протокола). Таким образом, когда возникает операция чтения, произойдет промах кэш-памяти, который вынуждает выполнить выборку новой копии данных. Для выполнения операции записи мы можем потребовать, чтобы процессор имел *достоверную (valid)* копию данных в своей кэш-памяти прежде, чем выполнять в нее запись. Таким образом, если оба процессора попытаются записать в один и тот же элемент данных одновременно, один из них выиграет состязание у второго (мы вскоре увидим, как принять решение, кто из них выиграет) и вызывает аннулирование его копии. Другой процессор для завершения своей операции записи должен сначала получить новую копию данных, которая теперь уже должна содержать обновленное значение.

Альтернативой протоколу записи с аннулированием является обновление всех копий элемента данных в случае записи в этот элемент данных. Этот тип протокола называется протоколом *записи с обновлением (write update protocol)* или протоколом *записи с трансляцией (write broadcast protocol).* Обычно в этом протоколе для снижения требований к полосе пропускания полезно отслеживать, является ли слово в кэш-памяти разделяемым объектом, или нет, а именно, содержится ли оно в других кэшах. Если нет, то нет никакой необходимости обновлять другой кэш или транслировать в него обновленные данные.

Разница в производительности между протоколами записи с обновлением и с аннулированием определяется тремя характеристиками:

1. 1. Несколько последовательных операций записи в одно и то же слово, не перемежающихся операциями чтения, требуют нескольких операций трансляции при использовании протокола записи с обновлением, но только одной начальной операции аннулирования при использовании протокола записи с аннулированием.
2. 2. При наличии многословных блоков в кэш-памяти каждое слово, записываемое в блок кэша, требует трансляции при использовании протокола записи с обновлением, в то время как только первая запись в любое слово блока нуждается в генерации операции аннулирования при использовании протокола записи с аннулированием. Протокол записи с аннулированием работает на уровне блоков кэш-памяти, в то время как протокол записи с обновлением должен работать на уровне отдельных слов (или байтов, если выполняется запись байта).
3. 3. Задержка между записью слова в одном процессоре и чтением записанного значения другим процессором обычно меньше при использовании схемы записи с обновлением, поскольку записанные данные немедленно транслируются в процессор, выполняющий чтение (предполагается, что этот процессор имеет копию данных). Для сравнения, при использовании протокола записи с аннулированием в процессоре, выполняющим чтение, сначала произойдет аннулирование его копии, затем будет производиться чтение данных и его приостановка до тех пор, пока обновленная копия блока не станет доступной и не вернется в процессор.

Эти две схемы во многом похожи на схемы работы кэш-памяти со сквозной записью и с записью с обратным копированием. Также как и схема отложенной записи с обратным копированием требует меньшей полосы пропускания памяти, так как она использует преимущества операций над целым блоком, протокол записи с аннулированием обычно требует менее тяжелого трафика, чем протокол записи с обновлением, поскольку несколько записей в один и тот же блок кэш-памяти не требуют трансляции каждой записи. При сквозной записи память обновляется почти мгновенно после записи (возможно с некоторой задержкой в буфере записи). Подобным образом при использовании протокола записи с обновлением другие копии обновляются так быстро, насколько это возможно. Наиболее важное отличие в производительности протоколов записи с аннулированием и с обновлением связано с характеристиками прикладных программ и с выбором размера блока.

Основы реализации

Ключевым моментом реализации в многопроцессорных системах с небольшим числом процессоров как схемы записи с аннулированием, так и схемы записи с обновлением данных, является использование для выполнения этих операций механизма шины. Для выполнения операции обновления или аннулирования процессор просто захватывает шину и транслирует по ней адрес, по которому должно производиться обновление или аннулирование данных. Все процессоры непрерывно наблюдают за шиной, контролируя появляющиеся на ней адреса. Процессоры проверяют не находится ли в их кэш-памяти адрес, появившийся на шине. Если это так, то соответствующие данные в кэше либо аннулируются, либо обновляются в зависимости от используемого протокола. Последовательный порядок обращений, присущий шине, обеспечивает также строго последовательное выполнение операций записи, поскольку когда два процессора конкурируют за выполнение записи в одну и ту же ячейку, один из них должен получить доступ к шине раньше другого. Один процессор, получив доступ к шине, вызовет необходимость обновления или аннулирования копий в других процессорах. В любом случае, все записи будут выполняться строго последовательно. Один из выводов, который следует сделать из анализа этой схемы, заключается в том, что запись в разделяемый элемент данных не может закончиться до тех пор, пока она не захватит доступ к шине.

В дополнение к аннулированию или обновлению соответствующих копий блока кэш-памяти, в который производилась запись, мы должны также разместить элемент данных, если при записи происходит промах кэш-памяти. В кэш-памяти со сквозной записью последнее значение элемента данных найти легко, поскольку все записываемые данные всегда посылаются также и в память, из которой последнее записанное значение элемента данных может быть выбрано (наличие буферов записи может привести к некоторому усложнению).

Однако для кэш-памяти с обратным копированием задача нахождения последнего значения элемента данных сложнее, поскольку это значение скорее всего находится в кэше, а не в памяти. В этом случае используется та же самая схема наблюдения, что и при записи: каждый процессор наблюдает и контролирует адреса, помещаемые на шину. Если процессор обнаруживает, что он имеет модифицированную ("грязную") копию блока кэш-памяти, то именно он должен обеспечить пересылку этого блока в ответ на запрос чтения и вызвать отмену обращения к основной памяти. Поскольку кэши с обратным копированием предъявляют меньшие требования к полосе пропускания памяти, они намного предпочтительнее в мультипроцессорах, несмотря на некоторое увеличение сложности. Поэтому далее мы рассмотрим вопросы реализации кэш-памяти с обратным копированием.

Для реализации процесса наблюдения могут быть использованы обычные теги кэша. Более того, упоминавшийся ранее *бит достоверности (valid bit),* позволяет легко реализовать аннулирование. Промахи операций чтения, вызванные либо аннулированием, либо каким-нибудь другим событием, также не сложны для понимания, поскольку они просто основаны на возможности наблюдения. Для операций записи мы хотели бы также знать, имеются ли другие кэшированные копии блока, поскольку в случае отсутствия таких копий, запись можно не посылать на шину, что сокращает время на выполнение записи, а также требуемую полосу пропускания.

Чтобы отследить, является ли блок разделяемым, мы можем ввести дополнительный бит состояния (shared), связанный с каждым блоком, точно также как это делалось для битов достоверности (valid) и модификации (modified или dirty) блока. Добавив бит состояния, определяющий является ли блок разделяемым, мы можем решить вопрос о том, должна ли запись генерировать операцию аннулирования в протоколе с аннулированием, или операцию трансляции при использовании протокола с обновлением. Если происходит запись в блок, находящийся в состоянии "разделяемый" при использовании протокола записи с аннулированием, кэш формирует на шине операцию аннулирования и помечает блок как *частный* (private). Никаких последующих операций аннулирования этого блока данный процессор посылать больше не будет. Процессор с *исключительной* (exclusive) копией блока кэш-памяти обычно называется "владельцем" (owner) блока кэш-памяти.

При использовании протокола записи с обновлением, если блок находится в состоянии "разделяемый", то каждая запись в этот блок должна транслироваться. В случае протокола с аннулированием, когда посылается операция аннулирования, состояние блока меняется с "разделяемый" на "неразделяемый" (или "частный"). Позже, если другой процессор запросит этот блок, состояние снова должно измениться на "разделяемый". Поскольку наш наблюдающий кэш видит также все промахи, он знает, когда этот блок кэша запрашивается другим процессором, и его состояние должно стать "разделяемый".

Поскольку любая транзакция на шине контролирует адресные теги кэша, потенциально это может приводить к конфликтам с обращениями к кэшу со стороны процессора. Число таких потенциальных конфликтов можно снизить применением одного из двух методов: дублированием тегов, или использованием многоуровневых кэшей с "охватом" (inclusion), в которых уровни, находящиеся ближе к процессору являются поднабором уровней, находящихся дальше от него. Если теги дублируются, то обращения процессора и наблюдение за шиной могут выполняться параллельно. Конечно, если при обращении процессора происходит промах, он должен будет выполнять арбитраж с механизмом наблюдения для обновления обоих наборов тегов. Точно также, если механизм наблюдения за шиной находит совпадающий тег, ему будет нужно проводить арбитраж и обращаться к обоим наборам тегов кэша (для выполнения аннулирования или обновления бита "разделяемый"), возможно также и к массиву данных в кэше, для нахождения копии блока. Таким образом, при использовании схемы дублирования тегов процессор должен приостановиться только в том случае, если он выполняет обращение к кэшу в тот же самый момент времени, когда механизм наблюдения обнаружил копию в кэше. Более того, активность механизма наблюдения задерживается только когда кэш имеет дело с промахом.

Наимено-вание	Тип протокола	Стратегия записи в память	Уникальные свойства	Применение
Одиночная запись	Запись с аннулированием	Обратное копирование при первой записи	Первый описанный в литературе протокол наблюдения	-
Synapse N+1	Запись с аннулированием	Обратное копирование	Точное состояние, где “владельцем является память”	Машины Synapse Первые машины с когерентной кэш-памятью
Berkely	Запись с аннулированием	Обратное копирование	Состояние “разделяемый”	Машина SPUR университета Berkely
Illinois	Запись с аннулированием	Обратное копирование	Состояние “приватный”; может передавать данные из	Серии Power и Challenge компании Silicon Graphics

			любого кэша	
“Firefly”	Запись с трансляцией	Обратное копирование для “приватных” блоков и сквозная запись для “разделяемых”	Обновление памяти во время трансляции	SPARCcenter 2000

Рис. 10.4. Примеры протоколов наблюдения

Если процессор использует многоуровневый кэш со свойствами охвата, тогда каждая строка в основном кэше имеется и во вторичном кэше. Таким образом, активность по наблюдению может быть связана с кэшем второго уровня, в то время как большинство активностей процессора может быть связано с первичным кэшем. Если механизм наблюдения получает попадание во вторичный кэш, тогда он должен выполнять арбитраж за первичный кэш, чтобы обновить состояние и возможно найти данные, что обычно будет приводить к приостановке процессора. Такое решение было принято во многих современных системах, поскольку многоуровневый кэш позволяет существенно снизить требований к полосе пропускания. Иногда может быть даже полезно дублировать теги во вторичном кэше, чтобы еще больше сократить количество конфликтов между активностями процессора и механизма наблюдения.

В реальных системах существует много вариаций схем когерентности кэша, в зависимости от того используется ли схема на основе аннулирования или обновления, построена ли кэш-память на принципах сквозной или обратной записи, когда происходит обновление, а также имеет ли место состояние "владения" и как оно реализуется. На рис. 10.4 представлены несколько протоколов с наблюдением и некоторые машины, которые используют эти протоколы.

6.7. **10.3. Многопроцессорные системы с локальной памятью и многомашинные системы**

Существуют два различных способа построения крупномасштабных систем с распределенной памятью. Простейший способ заключается в том, чтобы исключить аппаратные механизмы, обеспечивающие когерентность кэш-памяти, и сосредоточить внимание на создании масштабируемой системы памяти. Несколько компаний разработали такого типа машины. Наиболее известным примером такой системы является компьютер T3D компании Cray Research. В этих машинах память распределяется между узлами (процессорными элементами) и все узлы соединяются между собой посредством того или иного типа сети. Доступ к памяти может быть локальным или удаленным. Специальные контроллеры, размещаемые в узлах сети, могут на основе анализа адреса обращения принять решение о том, находятся ли требуемые данные в локальной памяти данного узла, или размещаются в памяти удаленного узла. В последнем случае контроллеру удаленной памяти посылается сообщение для обращения к требуемым данным.

Чтобы обойти проблемы когерентности, разделяемые (общие) данные не кэшируются. Конечно, с помощью программного обеспечения можно реализовать некоторую схему кэширования разделяемых данных путем их копирования из общего адресного пространства в локальную память конкретного узла. В этом случае когерентностью памяти также будет управлять программное обеспечение. Преимуществом такого подхода является практически минимальная необходимая поддержка со стороны аппаратуры, хотя наличие, например, таких возможностей как блочное (групповое)

копирование данных было бы весьма полезным. Недостатком такой организации является то, что механизмы программной поддержки когерентности подобного рода кэш-памяти компилятором весьма ограничены. Существующая в настоящее время методика в основном подходит для программ с хорошо структурированным параллелизмом на уровне программного цикла.
Машины с архитектурой, подобной Cray T3D, называют процессорами (машинами) с массовым параллелизмом (MPP Massively Parallel Processor). К машинам с массовым параллелизмом предъявляются взаимно исключающие требования. Чем больше объем устройства, тем большее число процессоров можно расположить в нем, тем длиннее каналы передачи управления и данных, а значит и меньше тактовая частота. Происшедшее возрастание нормы массивности для больших машин до 512 и даже 64К процессоров обусловлено не ростом размеров машины, а повышением степени интеграции схем, позволившей за последние годы резко повысить плотность размещения элементов в устройствах. Топология сети обмена между процессорами в такого рода системах может быть различной. На рис. 10.5 приведены характеристики сети обмена для некоторых коммерческих MPP.
Для построения крупномасштабных систем альтернативой рассмотренному в предыдущем разделе протоколу наблюдения может служить протокол на основе справочника, который отслеживает состояние кэшей. Такой подход предполагает, что логически единый справочник хранит состояние каждого блока памяти, который может кэшироваться. В справочнике обычно содержится информация о том, в каких кэшах имеются копии данного блока, модифицировался ли данный блок и т.д. В существующих реализациях этого направления справочник размещается рядом с памятью. Имеются также протоколы, в которых часть информации размещается в кэш-памяти. Положительной стороной хранения всей информации в едином справочнике является простота протокола, связанная с тем, что вся необходимая информация сосредоточена в одном месте. Недостатком такого рода справочников является его размер, который пропорционален общему объему памяти, а не размеру кэш-памяти. Это не составляет проблемы для машин, состоящих, например, из нескольких сотен процессоров, поскольку связанные с реализацией такого справочника накладные расходы можно преодолеть. Но для машин большего размера необходима методика, позволяющая эффективно масштабировать структуру справочника.

Фирма	Название	Количество узлов	Базовая топология	Разрядность связи (бит)	Частота синхронизации (Мгц)	Пиковая полоса пропускания связи (Мбайт/с)	Общая полоса пропускания (Мбайт/с)	Год выпуска
Thinking Machines	CM-2	1024-4096	12-мерный куб	1	7	1	1024	1987
nCube	nCube/ten	1-1024	10-мерный куб	1	10	1.2	640	1987
Intel	iPSC/2	16-128	7-мерный куб	1	16	2	345	1988
Maspar	MP-1216	32-512	2-мерная сеть+ступенчатая Omega	1	25	3	1300	1989

Intel	Delta	540	2-мерная сеть	16	40	40	640	1991
Thinking Machines	CM-5	32-2048	многосту-пенчатое толстое дерево	4	40	20	10240	1991
Meiko	CS-2	2-1024	многосту-пенчатое толстое дерево	8	70	50	50000	1992
Intel	Paragon	4-1024	2-мерная сеть	16	100	200	6400	1992
Cray Research	T3D	16-1024	3-мерный тор	16	150	300	19200	1993

Рис. 10.5. Характеристики межсоединений некоторых коммерческих MPP

В частности, чтобы предотвратить появление узкого горла в системе, связанного с единым справочником, можно распределить части этого справочника вместе с устройствами распределенной локальной памяти. Таким образом можно добиться того, что обращения к разным справочникам (частям единого справочника) могут выполняться параллельно, точно также как обращения к локальной памяти в распределенной памяти могут выполняться параллельно, существенно увеличивая общую полосу пропускания памяти. В распределенном справочнике сохраняется главное свойство подобных схем, заключающееся в том, что состояние любого разделяемого блока данных всегда находится во вполне определенном известном месте. На рис. 10.6 показан общий вид подобного рода машины с распределенной памятью. Вопросы детальной реализации протоколов когерентности памяти для таких машин выходят за рамки настоящего обзора.

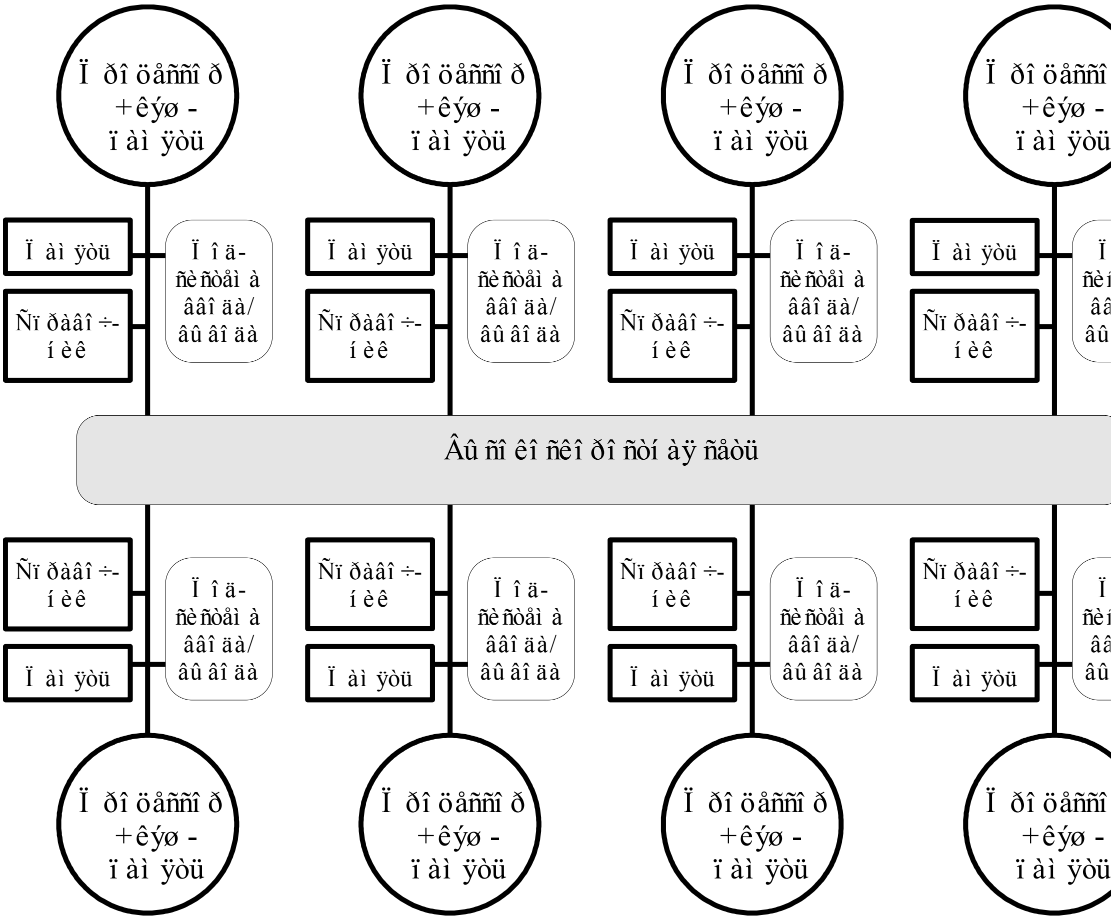


Рис. 10.6. Архитектура системы с распределенной внешней памятью и распределенным по узлам справочником

7. **11. Системы высокой готовности и отказоустойчивые системы**

7.1. **11.1. Основные определения**

Одной из основных проблем построения вычислительных систем во все времена остается задача обеспечения их продолжительного функционирования. Эта задача имеет три составляющих: *надежность*, *готовность* и *удобство обслуживания*. Все эти три составляющих предполагают, в первую очередь, борьбу с неисправностями системы, порождаемыми отказами и сбоями в ее работе. Эта борьба ведется по всем трем направлениям, которые взаимосвязаны и применяются совместно.

Повышение надежности основано на принципе предотвращения неисправностей путем снижения интенсивности отказов и сбоев за счет применения электронных схем и компонентов с высокой и сверхвысокой степенью интеграции, снижения уровня помех, облегченных режимов работы схем, обеспечения тепловых режимов их работы, а также за счет совершенствования методов сборки аппаратуры. Единицей измерения надежности является среднее время наработки на отказ (MTBF - Mean Time Between Failure).

Повышение готовности предполагает подавление в определенных пределах влияния отказов и сбоев на работу системы с помощью средств контроля и коррекции ошибок, а также средств автоматического восстановления вычислительного процесса после проявления неисправности, включая аппаратурную и программную избыточность, на основе которой реализуются различные варианты отказоустойчивых архитектур. Повышение готовности - есть способ борьбы за снижение времени простоя системы. Единицей измерения здесь является коэффициент готовности, который определяет вероятность пребывания системы в работоспособном состоянии в любой произвольный момент времени. Статистически коэффициент готовности определяется как MTBF/(MTBF+MTTR), где MTTR (Mean Time To Repair) - среднее время восстановления (ремонта), т.е. среднее время между моментом обнаружения неисправности и моментом возврата системы к полноценному функционированию.

Таким образом, основные эксплуатационные характеристики системы существенно зависят от удобства ее обслуживания, в частности от ремонтопригодности, контролепригодности и т.д.

В последние годы в литературе по вычислительной технике все чаще употребляется термин "системы высокой готовности", "системы высокой степени готовности", "системы с высоким коэффициентом готовности". Все эти термины по существу являются синонимами, однако как и многие термины в области вычислительной техники, термин "высокая готовность" понимается по-разному отдельными поставщиками и потребителями вычислительных систем. Совершенно аналогично, некоторые слова, связанные с термином "высокая готовность", такие, например, как "кластеризация", также употребляются в различных значениях. Важно иметь стандартный набор определений для того, чтобы предложения различных поставщиков можно было сравнивать между собой на основе одинаковых терминов.

Ниже приведены общепринятые в настоящее время определения, которые мы будем использовать для различных типов систем, свойством которых является та или иная форма снижения планового и непланового времени простоя:

- • *Высокая Готовность (High Availability).* Настоящие конструкции с высоким коэффициентом готовности для минимизации планового и непланового времени простоя используют обычную компьютерную технологию. При этом конфигурация системы обеспечивает ее быстрое восстановление после обнаружения неисправности, для чего в ряде мест используются избыточные аппаратные и программные средства. Длительность задержки, в течение которой программа, отдельный компонент или система простаивает, может находиться в диапазоне от нескольких секунд до нескольких часов, но более часто в диапазоне от 2 до 20 минут. Обычно системы высокой готовности хорошо масштабируются, предлагая пользователям большую гибкость, чем другие типы избыточности.
- • *Эластичность к отказам (Fault Resiliency).* Ряд поставщиков компьютерного оборудования делит весь диапазон систем высокой готовности на две части, при этом в верхней его части оказываются системы эластичные к отказам. Ключевым моментом в определении эластичности к отказам является более короткое время восстановления, которое позволяет системе быстро откатиться назад после обнаружения неисправности.
- • *Устойчивость к отказам (Fault Tolerance).* Отказоустойчивые системы имеют в своем составе избыточную аппаратуру для всех функциональных блоков, включая процессоры, источники питания, подсистемы ввода/вывода и подсистемы дисковой памяти. Если соответствующий функциональный блок неправильно функционирует, всегда имеется горячий резерв. В наиболее продвинутых отказоустойчивых системах избыточные аппаратные средства можно использовать для распараллеливания обычных работ. Время восстановления после обнаружения неисправности для переключения отказавших компонентов на избыточные для таких систем обычно меньше одной секунды.
- • *Непрерывная готовность (Continuous Availability).* Вершиной линии отказоустойчивых систем являются системы, обеспечивающие непрерывную готовность. Продукт с непрерывной готовностью, если он работает корректно, устраняет любое время простоя как плановое, так и неплановое. Разработка такой системы охватывает как аппаратные средства, так и программное обеспечение и позволяет проводить модернизацию (upgrade) и обслуживание в режиме on-line. Дополнительным требованием к таким системам является отсутствие деградации в случае отказа. Время восстановления после отказа не превышает одной секунды.
- • *Устойчивость к стихийным бедствиям (Disaster Tolerance).* Широкий ряд продуктов и услуг связан с обеспечением устойчивости к стихийным бедствиям. Иногда устойчивость к стихийным бедствиям рассматривается в контексте систем высокой готовности. Смысл этого термина в действительности означает возможность рестарта или продолжения операций на другой площадке, если основное месторасположение системы оказывается в нерабочем состоянии из-за наводнения, пожара или землетрясения. В простейшем случае, продукты, устойчивые к стихийным бедствиям, могут просто представлять собой резервные компьютеры, расположенные вне основного местоположения системы, сконфигурированные по спецификациям пользователя и доступные для использования в случае стихийного бедствия на основной площадке. В более

сложных случаях устойчивость к стихийным бедствиям может означать полное (зеркальное) дублирование системы вне основного местоположения, позволяющее принять на себя работу немедленно после отказа системы на основной площадке.

Все упомянутые типы систем высокой готовности имеют общую цель - минимизацию времени простоя. Имеется два типа времени простоя компьютера: плановое и неплановое. Минимизация каждого из них требует различной стратегии и технологии. Плановое время простоя обычно включает время, принятое руководством, для проведения работ по модернизации системы и для ее обслуживания. Неплановое время простоя является результатом отказа системы или компонента. Хотя системы высокой готовности возможно больше ассоциируются с минимизацией неплановых простоев, они оказываются также полезными для уменьшения планового времени простоя.

Возможно, наибольшим виновником планового времени простоя является резервное копирование данных. Некоторые конфигурации дисковых подсистем высокой готовности, особенно системы с зеркальными дисками, позволяют производить резервное копирование данных в режиме on-line. Следующим источником снижения планового времени простоя является организация работ по обновлению (модернизации) программного обеспечения. Сегодня некоторые отказоустойчивые системы и все системы с непрерывной готовностью позволяют производить модернизацию программного обеспечения в режиме on-line. Некоторые поставщики систем высокой готовности также обещают такие же возможности в течение ближайших нескольких лет.

В общем случае, неплановое время простоя, прежде всего, снижается за счет использования надежных частей, резервных магистралей или избыточного оборудования. Однако даже в этом случае система может требовать достаточно большого планового времени простоя.

Специальное программное обеспечение является существенной частью систем высокой готовности. При обнаружении неисправности системы оно обеспечивает управление конфигурацией аппаратных средств и программного обеспечения, а также в случае необходимости процедурами начальной установки, и перестраивает где надо структуры данных.

Высокая готовность не дается бесплатно. Общая стоимость подобных систем складывается из трех составляющих: начальной стоимости системы, издержек планирования и реализации, а также системных накладных расходов.

Для реализации системы высокой готовности пользователи должны в начале закупить собственно систему (или системы), включающую один или несколько процессоров в зависимости от требуемой вычислительной мощности и предполагаемой конфигурации, дополнительное программное обеспечение и дополнительное дисковое пространство.

Чтобы реализовать конфигурацию системы высокой готовности наиболее эффективным способом, особенно при использовании кластерных схем, требуется достаточно большое предварительное планирование. Например, чтобы иметь возможность переброски критичного приложения в случае отказа одного процессора на другой, пользователи должны определить, какие приложения являются наиболее критичными, проанализировать все возможные отказы и составить подробные планы восстановления на все случаи отказов.

Накладные расходы систем высокой готовности связаны с необходимостью поддержки довольно сложных программных продуктов, обеспечивающих высокую готовность. Для обеспечения дублирования записей на зеркальные диски в случае отсутствия специальных, предназначенных для этих целей процессоров, требуется поддержка дополнительной внешней памяти.

Стоимость системы высокой готовности в значительной степени зависит от выбранной конфигурации и ее возможностей. Ниже приведена некоторая информация, позволяющая грубо оценить различные типы избыточности.

Высокая Готовность. Дополнительная стоимость систем высокой готовности меняется в пределах от 10 до 100 процентов, обычно стремясь к середине этого диапазона. Дополнительная стоимость системы высокой готовности зависит от той степени, с которой пользователь способен использовать резервную систему для обработки своих приложений. Стоимость системы высокой готовности может реально превысить 100 процентов за счет программного обеспечения и необходимой начальной установки в случае применения резервной системы, которая не используется ни для чего другого. Однако обычно резервная система может быть использована для решения некритичных задач, значительно снижая стоимость.

Высокая эластичность. Дополнительная стоимость систем высокой эластичности к отказам, принадлежащих к верхнему уровню диапазона систем высокой готовности, лежит в пределах от 20 до 100 процентов, снова обычно стремясь к середине этого диапазона. Схемы высокой эластичности более сложны и предполагают более высокую стоимость планирования и большие накладные расходы, чем системы, принадлежащие нижнему уровню диапазона систем высокой готовности. В некоторых случаях однако, пользователь может в большей степени использовать общие процессорные ресурсы, тем самым уменьшая общую стоимость.

Непрерывная готовность. Надбавка к стоимости для систем с непрерывной готовностью находится в диапазоне от 20 до 100 или более процентов и обычно приближается к верхнему пределу этого диапазона. Программное обеспечение для обеспечения режима непрерывной готовности более сложное, чем для систем, обеспечивающих высокую эластичность к отказам. Большинство компонентов системы, такие как процессоры, источники питания, контроллеры и кабели, должны дублироваться, а иногда и троироваться. Пользователи систем непрерывной готовности, как и пользователи высоко эластичных к отказам систем, имеют возможность использовать весь набор ресурсов системы большую часть времени, по сравнению с пользователями более простых систем, принадлежащих нижнему уровню диапазона систем высокой готовности.

Устойчивость к стихийным бедствиям. Надбавка к стоимости для систем, устойчивых к стихийным бедствиям, может сильно варьироваться. С одной стороны, она может составлять, например, только несколько процентов стоимости системы при резервировании времени запасного компьютера, находящегося вне основной площадки. С другой стороны, стоимость системы может увеличиться в несколько раз, если необходимо обеспечить действительно быстрое переключение на другую систему, находящуюся на удаленной площадке с помощью высокоскоростных сетевых средств. Большинство предложений по системам, устойчивым к стихийным бедствиям, требуют существенного объема планирования.

Для того, чтобы снизить стоимость системы, следует тщательно оценивать действительно необходимый уровень готовности (т.е. осуществлять выбор между высокой готовностью, устойчивостью к отказам и/или устойчивостью к стихийным бедствиям) и вкладывать деньги только за обеспечение безопасности наиболее критичных для деятельности компании приложений и данных.

7.2. 11.2. Подсистемы внешней памяти высокой готовности

Первым шагом на пути обеспечения высокой готовности является защита наиболее важной части системы, а именно - данных. Разные типы конфигураций избыточной внешней памяти обеспечивают разную степень защиты данных и имеют разную стоимость.

Имеются три основных типа подсистем внешней памяти с высокой готовностью. Для своей реализации они используют технологию Избыточных Массивов Дешевых Дисков (RAID - Redundant Arrays of Inexpensive Disks). Наиболее часто используются следующие решения (более подробно об уровнях RAID см. разд. 9.3.2): RAID уровня 1 или зеркальные диски, RAID уровня 3 с четностью и RAID уровня 5 с распределенной четностью. Эти три типа внешней памяти в общем случае имеют практически почти мгновенное время восстановления в случае отказа. Кроме того, подобные устройства иногда позволяют пользователям смешивать и подбирать типы RAID в пределах одного дискового массива. В общем случае дисковые массивы представляются прикладной задаче как один диск.

Технология RAID уровня 1 (или зеркалирования дисков) основана на применении двух дисков так, что в случае отказа одного из них, для работы может быть использована копия, находящаяся на дополнительном диске. Программные средства поддержки зеркальных дисков обеспечивают запись всех данных на оба диска. Недостатком организации зеркальных дисков является удвоение стоимости аппаратных средств и незначительное увеличение времени записи, поскольку данные должны быть записаны на оба диска. Положительные стороны этого подхода включают возможность обеспечения резервного копирования в режиме on-line, а также замену дисков в режиме on-line, что существенно снижает плановое время простоя. Как правило, структура устройств с зеркальными дисками устраняет также единственность точки отказа, поскольку для подключения обоих дисков обычно предусматриваются два отдельных кабеля, а также два отдельных контроллера ввода/вывода.

В массивах RAID уровня 3 предусматривается использование одного дополнительного дискового накопителя, обеспечивающего хранение информации о четности (контрольной суммы) данных, записываемых на каждые два или четыре диска. Если один из дисков в массиве отказывает, информация о четности вместе с данными, находящимися на других оставшихся дисках, позволяет реконструировать данные, находившиеся на отказавшем накопителе.

Массив RAID уровня 5 является комбинацией RAID уровня 0, в котором данные расщепляются для записи на несколько дисков, и RAID уровня 3, в которых имеется один дополнительный диск. В RAID уровня 5 полезная информация четырех дисков и контрольная информация распределяется по всем пяти дискам так, что при отказе одного из них, оставшиеся четыре обеспечивают считывание необходимых данных. Методика расщепления данных позволяет также существенно увеличить скорость ввода/вывода при передаче больших объемов данных.

Диапазон возможных конструкций современных дисковых массивов достаточно широк. Он простираются от простых подсистем без многих дополнительных возможностей до весьма изощренных дисковых подсистем, которые позволяют пользователям смешивать и подбирать уровни RAID внутри одного устройства. Наиболее мощные дисковые подсистемы могут также содержать в своем составе процессоры, которые разгружают основную систему от выполнения рутинных операций ввода/вывода, форматирования дисков, защиты от ошибок и выполнения алгоритмов RAID. Большинство дисковых массивов снабжаются двумя портами, что позволяет пользователям подключать их к двум различным системам.

Добавка к стоимости дисковой подсистемы для организации зеркальных дисков стремится к 100%, поскольку требуемые диски должны дублироваться в избыточной конфигурации 1:1. Дополнительная стоимость дисков для RAID уровней 3 и 5 составляет либо 33% при наличии диска четности для каждых двух дисков, либо более часто 20% при наличии диска четности для каждых четырех накопителей. Кроме того, следует учитывать стоимость специального программного обеспечения, а также стоимость организации самого массива. Некоторые компании предлагают программные средства для организации зеркальных дисков при использовании обычных дисковых подсистем. Другие предлагают возможности зеркальной

организации в подсистемах RAID, специально изготовленных для этих целей. Подсистемы, использующие RAID уровней 3 и 5, отличаются по стоимости в зависимости от своих возможностей.
Реализация внешней памяти высокой готовности может приводить также к увеличению системных накладных расходов. Например, основной процессор системы вынужден обрабатывать две операции при каждой записи информации на зеркальные диски, если эти диски не являются частью зеркального дискового массива, который имеет свои собственные средства обработки. Однако наиболее сложные дисковые массивы позволяют снизить накладные расходы за счет использования процессоров ввода/вывода, являющихся частью аппаратуры дискового массива.
В настоящее время для устройств внешней памяти характерна тенденция уменьшения соотношения стоимости на единицу емкости памяти (1, 0.5 и менее долларов за мегабайт). Эта тенденция делает сегодняшние избыточные решения по внешней памяти даже менее дорогими, по сравнению со стоимостью подсистем с обычными дисками, выпускавшимися только год назад. Поэтому использование подсистемы RAID очень быстро становится одним из базовых требований обычных систем, а не специальным свойством систем высокой готовности.

7.3. 11.3. Требования, предъявляемые к системам высокой готовности

В настоящее время одним из ключевых требований пользователей UNIX-систем является возможность их наращивания с целью обеспечения более высокой степени готовности. Главными характеристиками систем высокой готовности по сравнению со стандартными системами являются пониженная частота отказов и более быстрый переход к нормальному режиму функционирования после возникновения неисправности посредством быстрого восстановления приложений и сетевых сессий до того состояния, в котором они находились в момент отказа системы. Следует отметить, что во многих случаях пользователей вполне может устроить даже небольшое время простоя в обмен на меньшую стоимость системы высокой готовности по сравнению со значительно более высокой стоимостью обеспечения режима непрерывной готовности.

7.3.1. 11.3.1. Конфигурации систем высокой готовности

Конфигурации систем высокой готовности, предлагаемые современной компьютерной промышленностью, простираются в широком диапазоне от "простейших" жестких схем, обеспечивающих дублирование основной системы отдельно стоящим горячим резервом в соотношении 1:1, до весьма свободных кластерных схем, позволяющих одной системе подхватить работу любой из нескольких систем в кластере в случае их неисправности.
Термин "кластеризация" на сегодня в компьютерной промышленности имеет много различных значений. Строгое определение могло бы звучать так: "реализация объединения машин, представляющегося единым целым для операционной системы, системного программного обеспечения, прикладных программ и пользователей". Машины, кластеризованные вместе таким способом могут при отказе одного процессора очень быстро перераспределить работу на другие процессоры внутри кластера. Это, возможно, наиболее важная задача многих поставщиков систем высокой готовности. Имеются несколько поставщиков, которые называют свои системы высокой готовности "кластерами" или "простыми кластерами", однако на сегодняшний день реально доступны только несколько кластеров, которые подпадают под строгое определение (см. ниже).
Современные конструкции систем высокой готовности предполагают использование горячего резерва (Fail-Over), включая переключение прикладных программ и

пользователей на другую машину с гарантией отсутствия потерь или искажений данных во время отказа и переключения. В зависимости от свойств системы, некоторые или все эти процессы могут быть автоматизированы.

Системы высокой готовности связаны со своими резервными системами посредством очень небольшого программного демона "сердечный пульс", который позволяет резервной системе управлять основной системой или системами, которые она резервирует. Когда "пульс" пропадает, кластер переходит в режим переключения на резервную систему.

Такое переключение может выполняться вручную или автоматически, и имеется несколько уровней автоматизации этого процесса. Например, в некоторых случаях пользователи инструктируются о том, что они должны выйти и снова войти в систему. В других случаях переключение осуществляется более прозрачным для пользователя способом: он только должен подождать в течение короткого периода времени. Иногда пользователь может делать выбор между ручным и автоматическим переключением. В некоторых системах пользователи могут продолжить работу после переключения именно с той точки, где они находились во время отказа. В других случаях их просят повторить последнюю транзакцию.

Резервная система не обязательно должна полностью повторять систему, которую она резервирует (конфигурации систем могут отличаться). Это позволяет в ряде случаев сэкономить деньги за счет резервирования большой системы или систем с помощью системы меньшего размера и предполагает либо снижение производительности в случае отказа основной системы, либо переключение на резервную систему только критичных для жизнедеятельности организации приложений.

Следует добавить, что одни пользователи предпочитают не выполнять никаких приложений на резервной машине, хотя другие наоборот стараются немного нагрузить резервный сервер в кластере. Возможность выбора конфигурации системы с помощью процедур начальной установки дает пользователям большую гибкость, позволяя постепенно использовать весь заложенный в системе потенциал.

7.3.2. 11.3.2. Требования начальной установки системы

Большинство систем высокой готовности требуют включения в свой состав процедур начальной установки (System Setup), обеспечивающих конфигурацию кластера для подобающего выполнения процедур переключения, необходимых в случае отказа. Пользователи могут запрограммировать "скрипты" начальной установки самостоятельно или попросить системного интегратора или поставщика проделать эту работу. В зависимости от того, насколько сложна начальная установка системы, и в зависимости от типа системы, с которой мигрирует пользователь, написание "скриптов", которые управляют действиями системы высокой готовности в случае отказа, может занять от одного - двух дней до нескольких недель или даже месяцев для опытных программистов. Многие поставщики обеспечивают несколько стандартных "скриптов" начальной установки. Кроме того, некоторые из них предоставляют сервисные услуги по начальной установке конфигурации, которые включают программирование сценариев переключения на горячий резерв в случае отказа, а также осуществляют работу с заказчиком по написанию или модификации "скриптов". Пользователи могут самостоятельно создавать "скрипты", однако для реализации подобающей конфигурации требуется высококвалифицированный программист - знаток UNIX и C.

Время простоя при переключении системы на резервную для систем высокой готовности может меняться в диапазоне от нескольких секунд до 20-40 и более минут. Процедура переключения на резерв включает в себя следующие этапы: резервная машина обнаруживает отказ основной и затем следует предписаниям скрипта, который вероятнее всего включает перезапуск системы, передачу адресов пользователей, получение и запуск необходимых приложений, а также выполнение определенных шагов по обеспечению корректного состояния данных. Время восстановления зависит главным образом от того, насколько быстро вторая машина сможет получить и запустить приложения, а также от того, насколько быстро операционная система и приложения, такие как базы данных или мониторы транзакций, смогут получить приведенные в порядок данные.

В общем случае аппаратное переключение на резерв занимает по порядку величины одну - две минуты, а система перезагружается за следующие одну - две минуты. В большинстве случаев от 5 до 20 минут требуется на то, чтобы получить и запустить приложение с полностью восстановленными данными. В противном случае пользователи инструктируются о необходимости заново ввести последнюю транзакцию.

Накладные системные расходы зависят от типа используемой системы и от сложности процедур ее начальной установки. Для простых процедур начальной установки при переходе на резерв они очень небольшие: от долей процента до 1.5%. Однако, чтобы получить истинную стоимость накладных расходов к этим накладным расходам необходимо добавить еще потери, связанные с недоиспользованием процессорной мощности резервной системы. Хотя покупатели стремятся использовать резервную систему для некритических приложений, она оказывается менее загруженной по сравнению с основной системой. Истинно кластерные системы, такие как VAXclasters компании DEC или кластер LifeKeeper Claster отделения NCR компании AT&T, являются примерами намного более сложного управления по сравнению с простыми процедурами начальной установки при переключении на резерв, и полностью используют все доступные процессоры. Однако организация таких систем влечет за собой и большие накладные расходы, которые увеличиваются с ростом числа узлов в кластере.

7.3.3. **11.3.3. Требования к системному программному обеспечению**

Поставщики, предлагающие системы с горячим резервом, обычно в своих версиях системы UNIX предоставляют также некоторые дополнительные свойства высокой готовности, обеспечивающие быструю загрузку и/или перезагрузку резервной системы в случае переключения при возникновении неисправности.

Журнализация файловой системы

Следствием журнализации изменений файловой системы является то, что файлы всегда находятся в готовом для использования состоянии. Когда система отказывает, журнализованная файловая система гарантирует, что файлы сохранены в последнем согласованном состоянии. Это позволяет осуществлять переключение на резервную систему без какой-либо порчи данных, а также либо вообще без каких-либо потерь данных, либо с потерей только одной последней транзакции. Такой подход отличается от систем, которые осуществляют журнализацию только метаданных файловой системы - процедура, которая помогает управлять целостностью файловой системы, но не целостностью данных.

Изоляция неисправного процесса

Для активно используемых компонентов программного обеспечения, таких как файловая система, часто применяется технология изоляции неисправных процессов, гарантирующая изоляцию ошибок в одной системе и невозможность их распространения за пределы этой системы.

Мониторы обработки транзакций

Иногда для управления переключением на резерв используются мониторы транзакций, гарантирующие отсутствие потерь данных. При этом для незавершенных транзакций может быть произведен откат назад, и базы данных возвращаются к известному согласованному состоянию. Для системы UNIX наиболее известными мониторами транзакций являются Tuxedo компании USL, Encina компании Transarc, CICS/6000 компании IBM и Top End компании NCR.

Другие функции программного обеспечения

В современных системах все возрастающую роль играет диагностика в режиме on-line, позволяющая предвосхищать проблемы, которые могут привести к простою системы. В настоящее время она специфична для каждой системы. В будущем, возможно, диагностика станет частью распределенного управления системой.

7.3.4. 11.3.4. Требования высокой готовности к прикладному программному обеспечению

Первыми открытыми системами, построенными в расчете на высокую готовность, были приложения баз данных и систем коммуникаций. Базы данных высокой готовности гарантируют непрерывный доступ к информации, которая является жизненно важной для функционирования многих корпораций и стержнем сегодняшней информационной экономики. Программное обеспечение систем коммуникаций высокой готовности усиливает средства горячего резервирования систем и составляет основу для распределенных систем высокой готовности и систем, устойчивых к стихийным бедствиям.

Высокая готовность баз данных. Несколько компаний, поставляющих базы данных, такие как Oracle, Sybase, Informix имеют в составе своих продуктов программное обеспечение, позволяющее выполнять быструю реконструкцию файлов в случае отказа системы. Это снижает время простоя для зеркальных серверов и кластерных решений.

Продукт Oracle7 Parallel Server позволяет нескольким системам одновременно работать с единым представлением базы данных Oracle. Балансировка нагрузки и распределенный менеджер блокировок, которые позволяют множеству систем обращаться к базе данных в одно и то же время, увеличивают потенциальную готовность базы данных.

Первоначально модель вычислительного кластера была разработана компанией DEC в конце 80-х годов. Она известна под названием VMS-кластеров (или VAX-кластеров), которые представляли собой объединенные в кластер миникомпьютеры VAX, работающие под управлением операционной системы VMS. Компания DEC была первым поставщиком Oracle Parallel Server на VMS-кластерах в 1991 году. После появления версии Oracle 7, у компании Oracle появился интерес к концепции базы данных на UNIX-кластерах, и в 1991 году она опубликовала программный интерфейс приложений (API) для своего менеджера блокировок. Публикация этого API означала открытое приглашение всех поставщиков вычислительных систем для организации поддержки на их платформах Oracle 7 Parallel Server. В результате основные поставщики UNIX-кластеров в настоящее время поддерживают этот продукт.

Кроме того, существуют продукты третьих фирм, дополняющие возможности баз данных известных производителей и обеспечивающие увеличение степени высокой готовности. Например, компания Afic Computer Incorporated (New York) продает продукт под названием Multi Server Option (MSO), который позволяет пользователям осуществлять через сеть TCP/IP одновременную запись в любое количество распределенных зеркальных баз данных Sybase. Информация распространяется широковещательным способом одновременно ко всем базам данных. MSO обеспечивает также балансировку нагрузки по множеству серверов. Если один из серверов отказывает, программное обеспечение перенаправит запрос на дублирующую базу данных в течение не более 30 секунд.

7.3.5. 11.3.5. Требования к сетевой организации и к коммуникациям

Системы высокой готовности требуют также высокой готовности коммуникаций. Чем быстрее коммуникации между машинами, тем быстрее происходит восстановление после отказа. Некоторые конфигурации систем высокой готовности имеют дублированные коммуникационные связи. В результате связь перестает быть точкой, отказ которой может привести к выходу из строя всей системы. Существующая в настоящее время тенденция в направлении сетей LAN и WAN с повышенной пропускной способностью обеспечивает как локальные, так и удаленные компьютеры более быстрыми коммуникациями, что приводит в итоге к более быстрому восстановлению в распределенных системах.

Современная сетевая технология сама по себе требует устранения таких точек, выход из строя которых, может привести к отказу всей сети. Сегодня при создании сетей характерно использование более сложных сетевых устройств от различных поставщиков, таких как маршрутизаторы и сетевые концентраторы (hub). Маршрутизаторы, которые определяют путь данных в сетях, могут вычислить новый путь в случае отказа связи. Концентраторы могут иметь конфигурации с избыточными устройствами и могут изолировать отказы в физической сети для предотвращения отказа всей сети. Важную роль в поддержании оптимального функционирования систем играют также сетевые анализаторы, позволяющие вызвать системного менеджера по любому симптому, который может потенциально привести к простою.

Высокая готовность сетевой организации всегда очень зависит от размера сети. По мере своего разрастания сеть стремится приобрести свойства встроенной надежности: чем больше сеть, тем более желательно использование альтернативных маршрутов, чтобы отказ одного компонента блокировал только какую-то небольшую ее часть.

В настоящее время ведутся активные разработки в направлении создания высокоскоростных сетей ATM. Широкое внедрение этой технологии позволит достаточно быстро передавать большие объемы информации через WAN и обеспечит высокоскоростное переключение в случае отказа между удаленными системами.

Цены на сети высокой готовности в значительной степени зависят от их конфигурации. Ниже даны несколько примеров одноранговых сетей высокой готовности.

Для небольшой сети, расположенной на одной площадке и имеющей менее 500 узлов, пользователь должен ожидать примерно 50-процентного увеличения стоимости за обеспечение высокой готовности.

Средняя по размеру сеть, предполагающая размещение на нескольких площадках и имеющая до 5000 узлов потребует от 30 до 40 процентов надбавки к стоимости обычной сети.

Для очень большой сети с более 5000 узлов, расположенных на одной площадке надбавка к стоимости за обеспечение высокой готовности уменьшается примерно до 20 - 30%.

В отказоустойчивых системах стоимость сети еще больше увеличивается. Надбавка к стоимости за базовую отказоустойчивую сеть обычно находится в пределах 80-100 и более процентов, и связана скорее не с размером сети, а с платой за высокую готовность.

7.4. 11.4. "Кластеризация" как способ обеспечения высокой готовности системы

7.4.1. 11.4.1. Базовая модель VAX/VMS кластеров

Компания DEC первой анонсировала концепцию кластерной системы в 1983 году, определив ее как группу объединенных между собой вычислительных машин, представляющих собой единый узел обработки информации. По существу VAX-кластер представляет собой слабосвязанную многомашинную систему с общей внешней памятью, обеспечивающую единый механизм управления и администрирования.

VAX-кластер обладает следующими свойствами:

Разделение ресурсов. Компьютеры VAX в кластере могут разделять доступ к общим ленточным и дисковым накопителям. Все компьютеры VAX в кластере могут обращаться к отдельным файлам данных как к локальным.

Высокая готовность. Если происходит отказ одного из VAX-компьютеров, задания его пользователей автоматически могут быть перенесены на другой компьютер кластера. Если в системе имеется несколько контроллеров HSC и один из них отказывает, другие контроллеры HSC автоматически подхватывают его работу.

Высокая пропускная способность. Ряд прикладных систем могут пользоваться возможностью параллельного выполнения заданий на нескольких компьютерах кластера.

Удобство обслуживания системы. Общие базы данных могут обслуживаться с единственного места. Прикладные программы могут инсталлироваться только однажды на общих дисках кластера и разделяться между всеми компьютерами кластера.

Расширяемость. Увеличение вычислительной мощности кластера достигается подключением к нему дополнительных VAX-компьютеров. Дополнительные накопители на магнитных дисках и магнитных лентах становятся доступными для всех компьютеров, входящих в кластер.

Работа VAX-кластера определяется двумя главными компонентами. Первым компонентом является высокоскоростной механизм связи, а вторым - системное программное обеспечение, которое обеспечивает клиентам прозрачный доступ к системному сервису. Физически связи внутри кластера реализуются с помощью трех различных шинных технологий с различными характеристиками производительности.

Основные методы связи в VAX-кластере представлены на рис. 11.1.

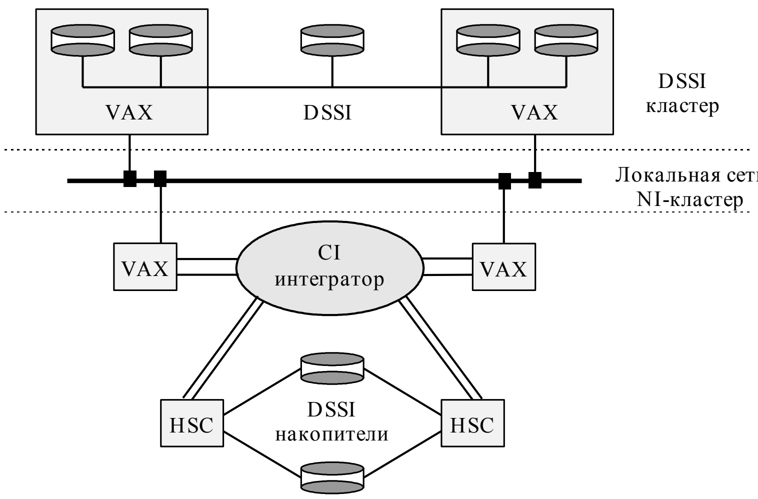


Рис. 11.1. VAX/VMS-кластер

Шина связи компьютеров CI (Computer Interconnect) работает со скоростью 70 Мбит/с и используется для соединения компьютеров VAX и контроллеров HSC с помощью коммутатора Star Coupler. Каждая связь CI имеет избыточные двойные линии, две для передачи и две для приема, используя базовую технологию CSMA, которая для устранения коллизий использует специфические для данного узла задержки. Максимальная длина связи CI составляет 45 метров. Звездообразный коммутатор Star Coupler может поддерживать подключение до 32 шин CI, каждая из которых предназначена для подсоединения компьютера VAX или контроллера HSC. Контроллер

HSC представляет собой интеллектуальное устройство, которое управляет работой дисковых и ленточных накопителей.
Компьютеры VAX могут объединяться в кластер также посредством локальной сети Ethernet, используя NI - Network Interconnect (так называемые локальные VAX-кластеры), однако производительность таких систем сравнительно низкая из-за необходимости делить пропускную способность сети Ethernet между компьютерами кластера и другими клиентами сети.
В начале 1992 года компания DEC анонсировала поддержку построения кластера на основе шины DSSI (Digital Storage System Interconnect). На шине DSSI могут объединяться до четырех компьютеров VAX нижнего и среднего класса. Каждый компьютер может поддерживать несколько адаптеров DSSI. Отдельная шина DSSI работает со скоростью 4 Мбайт/с (32 Мбит/с) и допускает подсоединение до 8 устройств. Поддерживаются следующие типы устройств: системный адаптер DSSI, дисковый контроллер серии RF и ленточный контроллер серии TF. DSSI ограничивает расстояние между узлами в кластере 25 метрами.
Во всем мире насчитывалось более 20000 установок VAX-кластеров. Почти все из них построены с использованием шинного интерфейса CI.
Системное программное обеспечение VAX-кластеров
Для гарантии правильного взаимодействия процессоров друг с другом при обращениях к общим ресурсам, таким, например, как диски, компания DEC использует распределенный менеджер блокировок DLM (Distributed Lock Manager). Очень важной функцией DLM является обеспечение когерентного состояния дисковых кэшей для операций ввода/вывода операционной системы и прикладных программ. Например, в приложениях реляционных СУБД DLM несет ответственность за поддержание согласованного состояния между буферами базы данных на различных компьютерах кластера.
Задача поддержания когерентности кэш-памяти ввода/вывода между процессорами в кластере подобна задаче поддержания когерентности кэш-памяти в сильно связанной многопроцессорной системе, построенной на базе некоторой шины. Блоки данных могут одновременно появляться в нескольких кэшах, и если один процессор модифицирует одну из этих копий, другие существующие копии не отражают уже текущее состояние блока данных. Концепция захвата блока (владения блоком) является одним из способов управления такими ситуациями. Прежде чем блок может быть модифицирован, должно быть обеспечено владение блоком.
Работа с DLM связана со значительными накладными расходами. Накладные расходы в среде VAX/VMS могут быть большими, требующими передачи до шести сообщений по шине CI для одной операции ввода/вывода. Накладные расходы могут достигать величины 20% для каждого процессора в кластере. Поставщики баз данных при использовании двухпроцессорного VAX-кластера обычно рассчитывают получить увеличение пропускной способности в 1.8 раза для транзакций выбора и в 1.3 раза для транзакций обновления базы данных.

7.4.2. **11.4.2. Критерии оценки кластеров Gartner Group**

Различные группы промышленных аналитиков считают VAX-кластеры образцом, по которому должны реализовываться другие кластерные системы, и примером свойств, которыми они должны обладать. VAX-кластеры являются очень надежными и высокопроизводительными вычислительными системами, основанными на миникомпьютерах. Хотя они базируются на патентованной операционной среде (VMS), опыт их эксплуатации показал, что основные свойства, которыми они обладают, очень хорошо соответствуют потребностям коммерческих учреждений.
В настоящее время на смену VAX-кластерам приходят UNIX-кластеры. При этом VAX-кластеры предлагают проверенный набор решений, который устанавливает критерии для оценки подобных систем. Одна из известных аналитических групп (Gartner Group)

опубликовала модель для сравнения, которая получила название "Критерии оценки кластеров" (MCS Research Note T-470-1411, November 22, 1993). Эта модель базируется на наборе свойств VAX-кластера и используется для сравнения UNIX-кластеров. Критерии этой модели представлены в табл. 11.1.

Таблица 11.1

Архитектурная модель	**VAX-кластер**
Масштабируемость	
Масштабируемость (>6 узлов)	да
Смешанный размер	да
Смешанные поколения	да
Единственное представление данных	да
Балансировка нагрузки	да
Общий планировщик работ	да
Общий менеджер систем	да
Надежность на уровне кластера	да
Готовность	
Подавление одиночного отказа или сбоя	да
Симметричный резерв аппаратных средств	да
Симметричный резерв программных средств	да
Автоматическая реконфигурация	
Автоматическое восстановление клиента	
Размещение на нескольких площадках	да
	да
	да

В настоящее время UNIX-кластеры все еще отстают от VAX-кластеров по функциональным возможностям. Одно из наибольших отличий связано с реализацией восстановления клиентов в случае отказа. В VAX-кластерах такое восстановление осуществляется средствами программного обеспечения самого VAX-кластера. В UNIX-кластерах эти возможности обычно реализуются отдельным уровнем программного обеспечения, называемым монитором транзакций. В UNIX-кластерах мониторы транзакций, кроме того, используются также для целей балансировки нагрузки, в то время как VAX-кластеры имеют встроенную в программное обеспечение утилиту балансировки загрузки.

Ожидается, что UNIX-кластеры подойдут и обгонят по функциональным возможностям VAX-кластеры в период 1996-1997 года.

7.4.3. 11.4.3. Кластеры Alpha/OSF компании DEC

Стратегическая линия новых продуктов компании DEC основана на новой аппаратной платформе Alpha AXP, которая поддерживает несколько операционных систем (в их числе OpenVMS, DEC OSF/1 и Windows NT). Компания объявила о создании UNIX-кластеров на своей новой платформе Alpha/OSF в 1993 году. Главной задачей при этом считается достижение тех же функциональных возможностей, которыми обладали VAX/VMS-кластеры.

В основу нового кластерного решения положен высокоскоростной коммутатор GigaSwitch, максимальная пропускная способность которого достигает 3.6 Гбайт/с. GigaSwitch представляет собой протокольно независимое устройство коммутации пакетов, которое может иметь 36 портов и поддерживать протоколы Ethernet, FDDI и ATM. Поддержка различных протоколов реализована с помощью набора специальных сменных карт. Предполагается, что к GigaSwitch помимо рабочих станций Alpha смогут подключаться и UNIX-станции других производителей.

В апреле 1994 года компания DEC анонсировала продукт под названием Integrated Cluster Server, который по своим функциональным характеристикам должен соответствовать стандарту VAX/VMS-кластеров. Основу этого решения составляет новая технология так называемой рефлективной памяти, имеющейся в каждом узле кластера. Устройства рефлективной памяти разных узлов кластера объединяются с помощью высокоскоростных каналов на основе интерфейса PCI (см. рис. 11.2). Обмен информацией между узлами типа "память-память" по этим каналам может производиться со скоростью 100 Мбайт/с, что на порядок превышает скорость обмена по обычным каналам ввода/вывода.

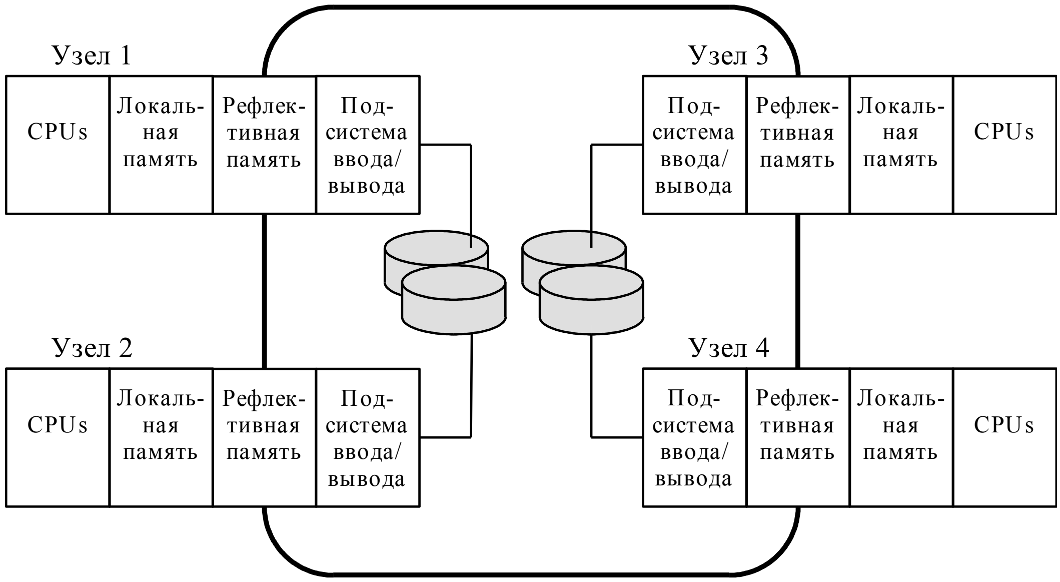


Рис. 11.2. Архитектура кластера компании DEC на базе рефлективной памяти и каналов "память-память"

Компания DEC выпускает RAID массивы StorageWorks, которые поддерживают RAID уровней 0, 1 и 5 и обеспечивают замену компонентов без выключения питания. В стоечной конфигурации поддерживается до 70 накопителей и дополнительные источники питания. Они могут выполнять до 400 операций ввода/вывода в секунду и подключаются к основным компьютерам через интерфейс Differential Fast SCSI-2. Скоростью интерфейса составляет 10 Мбайт/с, максимальная длина связи - 25 м.

7.4.4. **11.4.4. UNIX-кластеры компании IBM**

Компания IBM предлагает несколько типов слабо связанных систем на базе RS/6000, объединенных в кластеры и работающих под управлением программного продукта High-Availability Clastered Multiprocessor/6000 (HACMP/6000). В этих системах поддерживаются три режима автоматического восстановления системы после отказа:

Режим 1 - в конфигурации с двумя системами, одна из которых является основной, а другая находится в горячем резерве, в случае отказа обеспечивает автоматическое переключение с основной системы на резервную.

Режим 2 - в той же двухмашинной конфигурации позволяет резервному процессору обрабатывать некритичные приложения, выполнение которых в случае отказа основной системы можно либо прекратить совсем, либо продолжать их обработку в режиме деградации.

Режим 3 - можно действительно назвать кластерным решением, поскольку системы в этом режиме работают параллельно, разделяя доступ к логическим и физическим ресурсам пользуясь возможностями менеджера блокировок, входящего в состав HACMP/6000.

Начиная с объявления в 1991 году продукт HACMP/6000 постоянно развивался. В его состав были включены параллельный менеджер ресурсов, распределенный менеджер блокировок и параллельный менеджер логических томов, причем последний обеспечил возможность балансировки загрузки на уровне всего кластера. Максимальное количество узлов в кластере возросло до восьми. В настоящее время в составе кластера появились узлы с симметричной многопроцессорной обработкой, построенные по технологии Data Crossbar Switch, обеспечивающей линейный рост производительности с увеличением числа процессоров.

Первоначально обязательным требованием режима 3 было использование высокопроизводительной дисковой подсистемы IBM 9333, которая использовала последовательный дисковый интерфейс с поддержкой 17.6 Гбайт дискового пространства, а также дисковой подсистемы IBM 9334 с интерфейсом SCSI, обеспечивающим дисковое пространство в 8.2 Гбайт. В августе 1993 года была анонсирована первая подсистема RAID: две модели 7135 RAIDiant Array могут поддерживать до 768 Гбайт дискового пространства в стоечной конструкции серии 900 и 96 Гбайт в напольных тумбовых конструкциях, при этом реализуют RAID уровней 1, 3 и 5.

Кластеры RS/6000 строятся на базе локальных сетей Ethernet, Token Ring или FDDI и могут быть сконфигурированы различными способами с точки зрения обеспечения повышенной надежности:

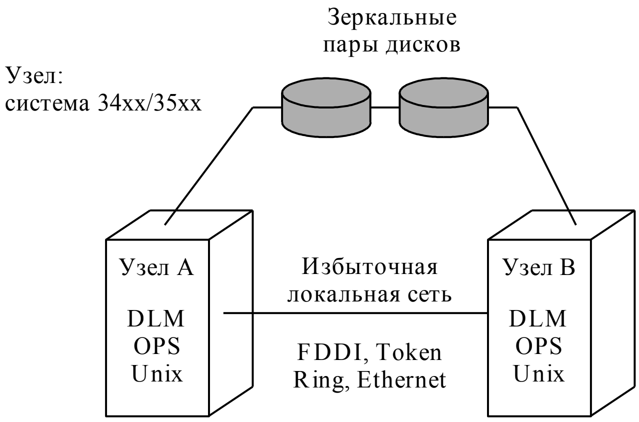
- • Горячий резерв или простое переключение в случае отказа. В этом режиме активный узел выполняет прикладные задачи, а резервный может выполнять некритичные задачи, которые могут быть остановлены в случае необходимости переключения при отказе активного узла.
- • Симметричный резерв. Этот режим аналогичен горячему резерву, но роли главного и резервного узлов не фиксированы.
- • Взаимный подхват или режим с распределением нагрузки. В этом режиме каждый узел в кластере может "подхватывать" задачи, которые выполняются на любом другом узле кластера.

7.4.5. **11.4.5. Кластеры AT&T GIS**

Отделение GIS (Global Information Systems) образовалось после покупки AT&T компании NCR, успешно работавшей в направлении создания систем с симметричной многопроцессорной обработкой (SMP) и систем с массовым параллелизмом (MPP) на базе микропроцессоров Intel. В 1993 году NCR анонсировала программное обеспечение для поддержки высокой готовности, получившее название LifeKeeper FRS (Fault Recilient Systems) Clastering Software, которое вместе с дисковыми массивами NCR позволяло строить высоконадежные кластерные решения. В состав кластеров NCR могут входить многопроцессорные системы серий 3400 и 3500, каждая из которых включает от 1 до 8 процессоров 486DX2 или Pentium (рис. 11.3).

Disk Array Subsystem 6298 включает до 20 дисковых накопителей емкостью 1 Гбайт и поддерживает RAID уровней 0, 1, 3 и 5 в любой комбинации. Подсистема обеспечивает замену дисковых накопителей, вентиляторов и источников питания в режиме on-line, т.е. без приостановки работы системы. В ней предусмотрено три порта и возможна поставка с избыточными контроллерами.

Программное обеспечение LifeKeeper допускает построение кластеров высокой готовности с четырьмя узлами, причем любой из узлов кластера может служить в качестве резервного для других узлов. Плановое время простоя для инсталляции программного обеспечения может быть значительно снижено, поскольку переключение на резерв можно инициировать вручную, затем модифицировать программное обеспечение и произвести обратное переключение с резерва. LifeKeeper обеспечивает также восстановление системы после обнаружения ошибок в системных, прикладных программах и периферийном оборудовании. Он обеспечивает автоматическое переключение при обнаружении отказа и инициируемое оператором обратное переключение. Все связи узлов кластера с помощью Ethernet, Token Ring и FDDI дублированы, а дисковые подсистемы, как уже отмечалось, могут подключаться сразу к нескольким узлам кластера. Все это обеспечивает построение системы, устойчивой к одиночным отказам, причем программное обеспечение выполняет автоматическое обнаружение отказов и восстановление системы.



- *Отказоустойчивый распределенный менеджер блокировок*
- *Средства автоматического восстановления на базе консольных сообщений*
- *Средства прозрачного восстановления сетевого адреса (горячее резервирование IP-соединений) или высокоуровневые средства восстановления клиентской части*
- *Средства обеспечения целостности данных на базе обобщенного SCSI-интерфейса*
- *Программные и аппаратные средства RAID*

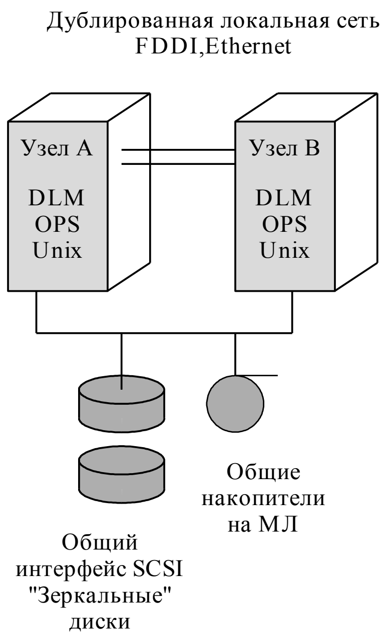
Рис. 11.3. Архитектура двухмашинного кластера AT&T GIS LifeKeeper FRS

При использовании Oracle Parallel Server распределенный менеджер блокировок, входящий в состав LifeKeeper, позволяет параллельной базе данных работать с системой высокой готовности.

В планы компании входит построение крупномасштабных кластеров для университетских кампусов, а также глобальных кластеров, способных продолжать работу в случае стихийных бедствий.

7.4.6. **11.4.6. Кластеры Sequent Computer Systems**

Sequent была, по-видимому, второй после IBM компанией, осуществившей поставки UNIX-кластеров баз данных в середине 1993 года. Она предлагает решения, соответствующие среднему и высокому уровню готовности своих систем. Первоначально Sequent Hi-Av Systems обеспечивали дублирование систем, которые разделяли общие диски. Пользователи могли выбирать ручной или автоматический режим переключения на резерв в случае отказа. Программный продукт ptx/CLASTERS, который может использоваться совместно с продуктом Hi-Av Systems, включает ядро, отказоустойчивый распределенный менеджер блокировок, обеспечивающий разделение данных между приложениями. Продукт ptx/NQS предназначен для балансировки пакетных заданий между узлами кластера, а ptx/LAT расширяет возможности управления пользовательскими приложениями в режиме on-line. Продукт ptx/ARGUS обеспечивает централизованное управление узлами кластера, а ptx/SVM (распределенный менеджер томов) представляет собой инструментальное средство управления внешней памятью системы. Hi-Av Systems обеспечивает также горячее резервирование IP адресов и позволяет кластеру, в состав которого входят до четырех узлов, иметь единственный сетевой адрес (рис. 11.4).



- *Отказоустойчивый распределенный менеджер блокировок*
- *Система автоматического восстановления на базе консольных сообщений*
- *Общие диски и ленты на базе контроллеров Fast-Wide SCSI-2*
- *Инструментальные средства третьих фирм для распределения заданий и формирования отчетов*
- *Инструментальные средства Sequent для администрирования, резервного копирования и управления томами в режиме on-line*
- *Поддержка монитора транзакций*

Рис. 11.4. Архитектура двухмашинного кластера SE90 компании Sequent

Компания Sequent одной из первых освоила технологию Fast-Wide SCSI, что позволило ей добиться значительного увеличения производительности систем при обработке транзакций. Компания поддерживает дисковые подсистемы RAID уровней 1, 3 и 5. Кроме того, она предлагает в качестве разделяемого ресурса ленточные накопители SCSI. Модель SE90 поддерживает кластеры, в состав которых могут входить два, три или четыре узла, представляющих собой многопроцессорные системы Symmetry 2000 или Symmetry 5000 в любой комбинации. Это достаточно мощные системы. Например, Sequent Symmetry 5000 Series 790 может иметь от 2 до 30 процессоров Pentium 66 МГц, оперативную память емкостью до 2 Гбайт и дисковую память емкостью до 840 Гбайт.
При работе с Oracle Parallel Server все узлы кластера работают с единственной копией базы данных, расположенной на общих разделяемых дисках.

7.4.7. **11.4.7. Системы высокой готовности Hewlett-Packard**

Продукты высокой готовности HP включают дисковые массивы, программное обеспечение Switchover/UX и SharePlex, а также заказные услуги по организации систем, устойчивых к стихийным бедствиям. HP до настоящего времени не поддерживает Oracle Parallel Server, хотя имеются планы по организации такой поддержки. Схема "слабо связанного" кластера HP включает до 8 серверов HP 9000 Series 800, причем семь из них являются основными системами, а восьмая находится в горячем резерве и готова заменить любую из основных систем в случае отказа (рис. 11.5). Для этого в нормальном режиме работы основные системы посылают резервной

сообщения (так называемый "пульс"), подтверждающие их работоспособное состояние. Если резервная система обнаруживает потерю "пульса" какой-либо основной системой, она прекращает выполнение своих процессов, берет на себя управление дисками отказавшей системы, осуществляет перезагрузку, переключает на себя сетевой адрес отказавшей системы и затем перезапускает приложения. Весь процесс переключения может занимать от 10 до 20 или более минут в зависимости от приложения.
Продукты Switchover и Shareplex могут использоваться совместно. При этом Switchover обеспечивает высокую готовность, а Shareplex расширяет возможности кластерной системы, предоставляя поддержку общего управления системами, отказоустойчивости при стихийных бедствиях и разделяемого доступа к распределенным ресурсам систем.

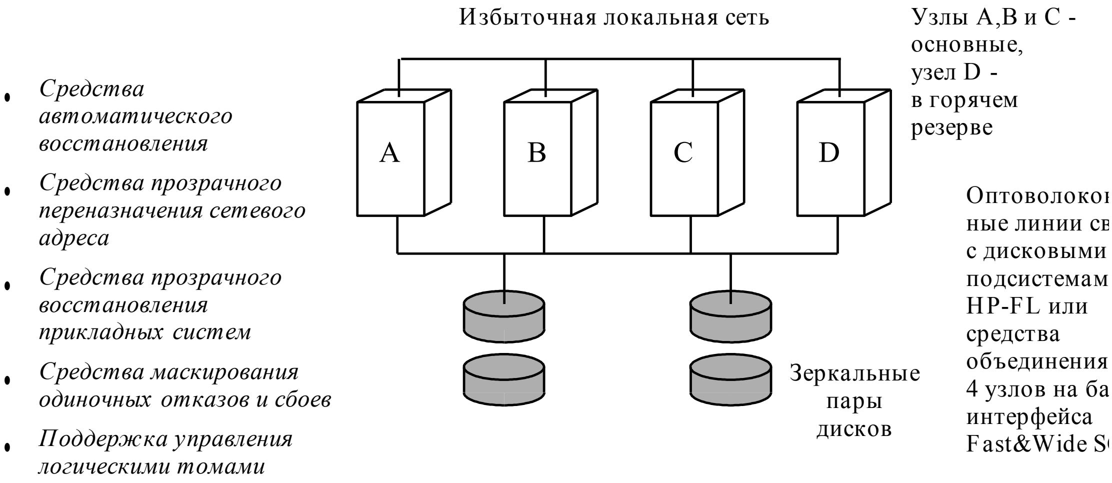


Рис. 11.5. Отказоустойчивая архитектура Switchover/UX компании Hewlett Packard

Хотя Oracle Parallel Server еще не поставлялся в составе кластерных систем HP, компания проводит активные работы в этом направлении. В частности, в последнее время появились новые программные продукты: MC/LockManager - менеджер блокировок, обеспечивающий связь между экземплярами Oracle, функционирующими в кластере, и MC/ServiceGuard - продукт, заменяющий ранее поставлявшийся Switchover.
Продукты HP известны своей высокой надежностью и хорошими показателями наработки на отказ. Дисковые массивы HP-FL поддерживают RAID уровней 0, 3 и 5, а также замену без выключения питания. В настоящее время в системах HP используются устройства Fast/Wide SCSI Disk Array, поддерживающие RAID уровней 0, 1, 3 и 5 с 50 Гбайт общего дискового пространства. В отличие от устройств HP-FL, подсоединение которых осуществлялось с помощью оптоволоконного кабеля, SCSI Disk Array подключается посредством обычных медных кабелей, длина которых может достигать 25 метров.
Новая версия ОС Version 10.0 HP-UX обеспечивает журнализацию файлов и распределенное управление блокировками, что позволяет HP активно конкурировать с продукцией высокой готовности других поставщиков.

7.4.8. **11.4.8. Кластерные решения Sun Microsystems**

Sun Microsystems предлагает кластерные решения на основе своего продукта SPARCclaster PDB Server, в котором в качестве узлов используются многопроцессорные SMP-серверы SPARCserver 1000 и SPARCcenter 2000.

Максимально в состав SPARCserver 1000 могут входить до восьми процессоров, а в SPARCcenter 2000 до 20 процессоров SuperSPARC. В комплект базовой поставки входят следующие компоненты: два кластерных узла на основе SPARCserver 1000/1000E или SPARCcenter 2000/2000E, два дисковых массива SPARCstorage Array, а также пакет средств для построения кластера, включающий дублированное оборудование для осуществления связи, консоль управления кластером Claster Management Console, программное обеспечение SPARCclaster PDB Software и пакет сервисной поддержки кластера.
Для обеспечения высокой производительности и готовности коммуникаций кластер поддерживает полное дублирование всех магистралей данных. Узлы кластера объединяются с помощью каналов SunFastEthernet с пропускной способностью 100 Мбит/с. Для подключения дисковых подсистем используется оптоволоконный интерфейс Fibre Channel с пропускной способностью 25 Мбит/с, допускающий удаление накопителей и узлов друг от друга на расстояние до 2 км. Все связи между узлами, узлами и дисковыми подсистемами дублированы на аппаратном уровне. Аппаратные, программные и сетевые средства кластера обеспечивают отсутствие такого места в системе, одиночный отказ или сбой которого выводил бы всю систему из строя.
SPARCclaster PDB Server поддерживает полностью автоматическое обнаружение отказов, их изоляцию и восстановление после отказа, причем обнаружение и изоляция отказа выполняются на разных уровнях в зависимости от отказавшего компонента (системы связи между узлами, дисковой подсистемы, сетевого подключения или целиком узла). При этом процесс восстановления осуществляется достаточно быстро, поскольку в случае подобных отказов не требуется полной перезагрузки системы.
В состав программного обеспечения кластера входят четыре основных компонента: отказоустойчивый распределенный менеджер блокировок, распределенный менеджер томов, программные средства управления обнаружением отказов и управления восстановлением, программное обеспечение управления кластером.
Sun Microsystems выпускает дисковые массивы, обеспечивающие RAID уровней 0 и 1. Ожидается появление поддержки RAID уровня 5. Максимальная емкость дискового массива для SPARCserver 1000 составляет 63 Гбайт при использовании SPARCstorage Array Model 100 и 324 Гбайт при использовании SPARCstorage Array Model 200. Для SPARCcenter 2000 эти цифры составляют соответственно 105 Гбайт и 324 Гбайт.

7.4.9. 11.4.9. Отказоустойчивые решения Data General

Линия продуктов высокой готовности компании Data General включает системы, основными свойствами которых являются: полностью автоматическое восстановление без потери данных, конфигурируемые дисковые и ленточные массивы и средства поддержания высокой готовности, встроенные в системное программное обеспечение.
Data General поставляет многопроцессорные SMP-серверы серий AV 5500, AV 8500 и AV 9500. Эти серверы поддерживают работу с отказоустойчивыми дисковыми и ленточными подсистемами CLARiiON, средства автоматической диагностики AV/Alert, инициируемые оператором или автоматически средства переключения на резервную систему, управление внешней памятью в режиме on-line, управление вводом/выводом и быстрое восстановление файлов.
При обнаружении отказа процессора, памяти или компоненты ввода/вывода система автоматически начинает процесс выключения и затем осуществляет собственную перезагрузку с исключением отказавших компонент. Стандартным средством указанных систем является наличие избыточных источников питания.
Максимальная степень готовности достигается при подключении двух серверов к высоконадежному дисковому массиву CLARiiON. Дисковые массивы CLARiiON Series C2000 Disk Array обеспечивают RAID уровней 0, 1, 3 и 5 в любых сочетаниях, до 20 накопителей в одном шасси общей емкостью до 80 Гбайт и возможность замены

накопителя без выключения питания. В конструкции дискового массива используются избыточные интеллектуальные контроллеры с дублированными связями, обеспечивающие отказоустойчивость. Ленточный массив CLARiiON Series 4000 поддерживает отказоустойчивое резервное копирование и восстановление. В составе массива используется специальный процессор, реализующий схему расщепления данных, подобную RAID уровня 5. Ленточный массив обеспечивает не только высокую пропускную способность, но и реализует защиту от отказов носителя и накопителя. В действительности, даже при отказе накопителя или картриджа, операции по резервному копированию или восстановлению данных продолжаются без потери данных. В массив можно устанавливать 3, 5 или 7 накопителей. При двухкратной компрессии данных общая емкость ленточного массива может достигать 48 Гбайт.

8. 12. Технические характеристики современных серверов

8.1. 12.1. Симметричные мультипроцессорные системы компании Bull

Группа компаний, объединенных под общим названием Bull, является одним из крупнейших производителей информационных систем на мировом компьютерном рынке и имеет свои отделения в Европе и США. Большая часть акций компании принадлежит французскому правительству. В связи с происходившей в последнем пятилетии перестройкой структуры компьютерного рынка компания объявила о своей приверженности к направлению построения открытых систем. В настоящее время компания продолжает выпускать компьютеры класса мейнфрейм (серии DPS9000/900, DPS9000/800, DPS9000/500) и среднего класса (серии DPS7000 и DPS6000), работающие под управлением фирменной операционной системы GCOS8, UNIX-системы (серии DPX/20, Escala), а также широкий ряд персональных компьютеров компании Zenith Data Systems (ZDS), входящей в группу Bull.

Активность Bull в области открытых систем сосредоточена главным образом на построении UNIX-систем. В результате технологического соглашения с компанией IBM, в 1992 году Bull анонсировала ряд компьютеров DPX/20, базирующихся на архитектуре POWER, а позднее в 1993 году на архитектуре PowerPC и работающих под управлением операционной системы AIX (версия системы UNIX компании IBM). Версия ОС AIX 4.1, разработанная совместно специалистами IBM и Bull, поддерживает симметричную многопроцессорную обработку.

Архитектура PowerScale, представляет собой первую реализацию симметричной мультипроцессорной архитектуры (SMP), разработанной Bull специально для процессоров PowerPC. В начале она была реализована на процессоре PowerPC 601, но легко модернизируется для процессоров 604 и 620. Эта новая SMP-архитектура используется в семействе систем Escala.

8.1.1. 12.1.1. Архитектура процессоров PowerPC

Основой архитектуры PowerPC является многокристальная архитектура POWER, которая была разработана, прежде всего, в расчете на однопроцессорную реализацию процессора. При разработке PowerPC для удовлетворения потребностей трех различных компаний (Apple, IBM и Motorola) в архитектуре POWER было сделано несколько изменений в следующих направлениях:

- • упрощение архитектуры с целью ее приспособления для реализации дешевых однокристальных процессоров;
- • устранение команд, которые могут стать препятствием повышения тактовой частоты;
- • устранение архитектурных препятствий суперскалярной обработке и внеочередному выполнению команд;
- • добавление свойств, необходимых для поддержки симметричной мультипроцессорной обработки;

- • добавление новых свойств, считающихся необходимыми для будущих прикладных программ;
- • обеспечение длительного времени жизни архитектуры путем ее расширения до 64-битовой.

Архитектура PowerPC поддерживает ту же самую базовую модель программирования и назначение кодов операций команд, что и архитектура POWER. В тех местах, где были сделаны изменения, которые могли потенциально нарушить двоичную совместимость с приложениями, написанными для архитектуры POWER, были расставлены "ловушки", обеспечивающие прерывание и эмуляцию с помощью программных средств. Такие изменения вводились, естественно, только в тех случаях, если соответствующая возможность либо использовалась не очень часто в кодах прикладных программ, либо была изолирована в библиотечных программах, которые можно просто заменить.

Микропроцессор PowerPC поддерживает мультипроцессорную обработку, в частности, модель тесно связанных вычислений в разделяемой (общей) памяти. Работа тесно связанных процессоров предполагает использование разными процессорами одной общей памяти и одной операционной системы, которая управляет всеми процессорами и аппаратурой системы. Процессоры должны конкурировать за разделяемые ресурсы.

В симметричной мультипроцессорной системе все процессоры считаются функционально эквивалентными и могут выполнять операции ввода/вывода и другие вычисления. Возможности управления подобной системой с разделяемой памятью реализованы в ОС AIX 4.1.

Разработанное Bull семейство Escala обеспечивает масштабируемость и высокую готовность систем, центральным местом которых является симметричная мультипроцессорная архитектура, названная PowerScale, позволяющая производить постепенную модернизацию и объединять в системе от 1 до 8 процессоров.

8.1.2. 12.1.2. Проблемы реализации SMP-архитектуры

По определению симметричная мультипроцессорная обработка (SMP) является архитектурой, в которой несколько процессоров разделяют доступ к единственной общей памяти и работают под управлением одной копии операционной системы. В этом случае задания могут соответствующим образом планироваться для работы на разных процессорах в пределах "пула имеющихся ресурсов", допуская распараллеливание, поскольку несколько процессов в такой системе могут выполняться одновременно.

Главным преимуществом архитектуры SMP по сравнению с другими подходами к реализации мультипроцессорных систем является прозрачность для программных приложений. Этот фактор существенно улучшает время выхода на рынок и готовность традиционных коммерческих приложений на системах SMP по сравнению с другими мультипроцессорными архитектурами.

В современных системах SMP наиболее актуальным вопросом разработки является создание высокопроизводительной подсистемы памяти для обеспечения высокоскоростных RISC-процессоров данными и командами. Общее решение этой проблемы заключается в использовании большой высокоскоросной кэш-памяти, т.е. в создании иерархии памяти между процессорами и разделяемой глобальной памятью. Архитектура PowerScale предлагает новый подход к решению вопросов традиционного узкого горла, ограничивающего производительность SMP-систем, а именно, новую организацию управления кэш-памятью и доступа к памяти.

PowerScale представляет собой высоко оптимизированную разработку, которая является результатом интенсивных исследований параметров производительности современных коммерческих приложений. Обычно выполнение этих прикладных систем связано с необходимостью манипулирования огромными объемами данных и разделения доступа к этим данным между многими пользователями или программами. Такого рода рабочая нагрузка характеризуется наличием больших рабочих наборов данных с низким уровнем локализации. При моделировании прикладных систем подобного профиля на системах SMP, были замечены два особых эффекта:

- • Из-за малой вероятности нахождения соответствующих данных в кэш-памяти возникает весьма интенсивный трафик между системной памятью и кэшами ЦП. Поэтому очень важно сконструировать систему, обеспечивающую широкополосный доступ к памяти.
- • В традиционной SMP-системе по умолчанию одна из задач планировщика заключается в том, чтобы запустить следующий разрешенный для выполнения процесс на первом же процессоре, который становится свободным. Поэтому по мере того, как увеличивается число процессоров и процессов, вероятность перемещения процессов с одного процессора на другой, также увеличивается. Эта побочная миграция процессов приводит к существенному увеличению уровня трафика между кэшами ЦП. Поэтому ключевым вопросом обеспечения высокой системной производительности становится физическая реализация когерентности кэш-памяти.

В традиционной SMP-архитектуре связи между кэшами ЦП и глобальной памятью реализуются с помощью общей шины памяти, разделяемой между различными процессорами. Как правило, эта шина становится слабым местом конструкции системы и стремится к насыщению при увеличении числа инсталлированных процессоров. Это происходит потому, что увеличивается трафик пересылок между кэшами и памятью, а также между кэшами разных процессоров, которые конкурируют между собой за пропускную способность шины памяти. При рабочей нагрузке, характеризующейся интенсивной обработкой транзакций, эта проблема является даже еще более острой.

В архитектуре PowerScale компании Bull интерфейс памяти реализован с учетом указанного выше профиля приложений и рассчитан на использование нескольких поколений процессоров со все возрастающей производительностью. В действительности архитектура PowerScale с самого начала была разработана в расчете на поддержку до 8 процессоров PowerPC 620.

8.1.3. 12.1.3. Описание архитектуры PowerScale

В архитектуре PowerScale (рис. 12.1) основным средством оптимизации доступа к разделяемой основной памяти является использование достаточно сложной системной шины. В действительности эта "шина" представляет собой комбинацию шины адреса/управления, реализованной классическим способом, и набора магистралей данных, которые соединяются между собой посредством высокоскоростного матричного коммутатора. Эта система межсоединений получила название MPB_SysBus. Шина памяти используется только для пересылки простых адресных тегов, а неблокируемый матричный коммутатор - для обеспечения более интенсивного трафика данных. К матричному коммутатору могут быть подсоединены до 4 двухпроцессорных портов, порт ввода/вывода и подсистема памяти.

Главным преимуществом такого подхода является то, что он позволяет каждому процессору иметь прямой доступ к подсистеме памяти. Другим важным свойством реализации является использование расслоения памяти, что позволяет многим процессорам обращаться к памяти одновременно.

Ниже приведена схема, иллюстрирующая общую организацию доступа к памяти (рис. 12.2) Каждый процессорный модуль имеет свой собственный выделенный порт памяти для пересылки данных. При этом общая шина адреса и управления гарантирует, что на уровне системы все адреса являются когерентными.

Вопросы балансировки нагрузки

В процессе разработки системы был сделан выбор в направлении использования больших кэшей второго уровня (L2), дополняющих кэши первого уровня (L1), интегрированные в процессорах PowerPC. Это решение породило необходимость более глубокого рассмотрения работы системы в целом. Чтобы получить полное преимущество от использования пространственной локальности данных, присутствующих в кэшах L2, необходимо иметь возможность снова назначить ранее отложенный процесс на процессор, на котором он ранее выполнялся, даже если этот процессор не является следующим свободным процессором. Подобная "настройка" системы является основой для балансировки нагрузки и аналогична процессу планирования.

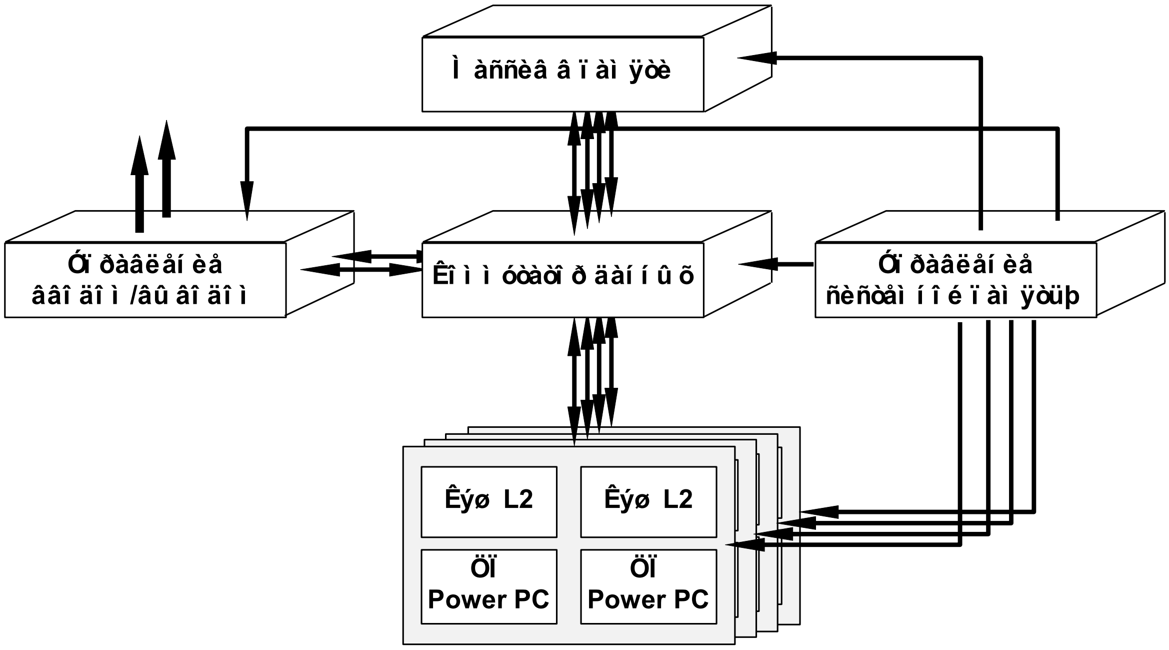


Рис. 12.1. Архитектура PowerScale

Очевидно, что всегда полезно выполнять процесс на одном и том же процессоре и иметь более высокий коэффициент попаданий в кэш, чем при выполнении процесса на следующем доступном процессоре. Используя алгоритмы, базирующиеся на средствах ядра системы, можно определить наиболее подходящее использование пула процессоров с учетом текущего коэффициента попаданий в кэш. Это позволяет оптимизировать уровень миграции процессов между процессорами и увеличивает общую пропускную способность системы.

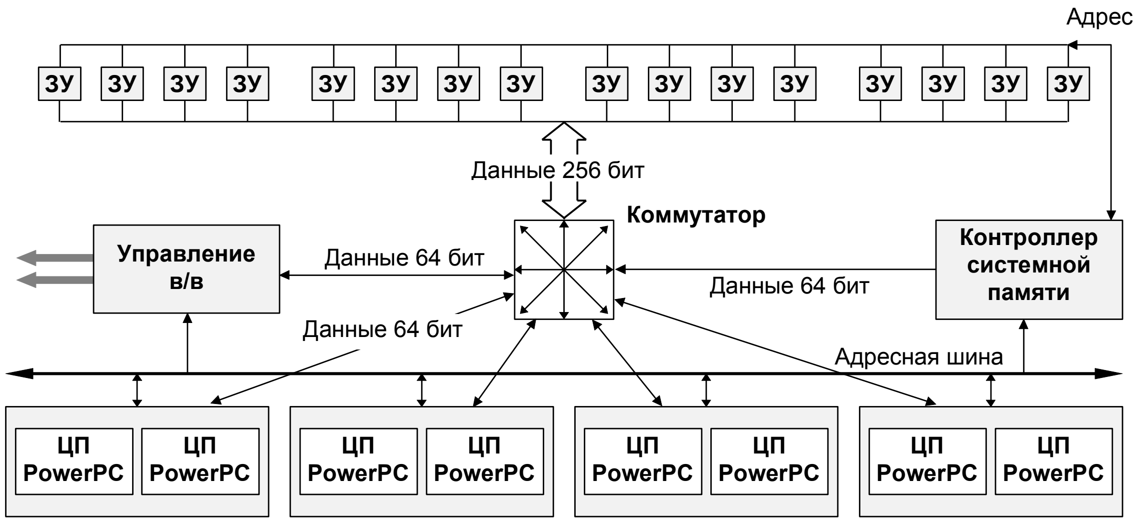


Рис. 12.2. Схема организации доступа к памяти

Модель памяти

Процессор PowerPC определяет слабо упорядоченную модель памяти, которая позволяет оптимизировать использование пропускной способности памяти системы. Это достигается за счет того, что аппаратуре разрешается переупорядочивать операции загрузки и записи так, что требующие длительного времени операции загрузки могут выполняться ранее определенных операций записи. Такой подход позволяет уменьшить действительную задержку операций загрузки. Архитектура PowerScale полностью поддерживает эту модель памяти как на уровне процессора за счет набора команд PowerPC, так и глобально путем реализации следующих ограничений:

- • Обращения к глобальным переменным синхронизации выполняются строго последовательно.
- • Никакое обращение к переменной синхронизации не выдается процессором до завершения выполнения всех обращений к глобальным данным.
- • Никакие обращения к глобальным данным не выдаются процессором до завершения выполнения предыдущих обращений к переменной синхронизации.

Для обеспечения подобной модели упорядоченных обращений к памяти на уровне каждого процессора системы используются определенная аппаратная поддержка и явные команды синхронизации. Кроме того, на системном уровне соблюдение необходимых протоколов для обеспечения упорядочивания обращений между процессорами или между процессорами и подсистемой ввода/вывода возложено на программное обеспечение.

Подсистема памяти

С реализацией архитектуры глобальной памяти в мультипроцесорной системе обычно связан очень важный вопрос. Как объединить преимущества "логически" локальной для каждого процессора памяти, имеющей малую задержку доступа, с требованиями реализации разделяемой глобальной памяти?

Компания Bull разработала патентованную архитектуру, в которой массив памяти полностью расслоен до уровня длины строки системного кэша (32 байта). Такая организация обеспечивает минимум конфликтов между процессорами при работе подсистемы памяти и гарантирует минимальную задержку. Требование реализации глобальной памяти обеспечивается тем, что массив памяти для программных средств всегда представляется непрерывным.

Предложенная конструкция решает также проблему, часто возникающую в других системах, в которых использование методов расслоения для организации последовательного доступа к различным областям памяти возможно только, если платы памяти устанавливаются сбалансировано. Этот, кажущийся тривиальным, вопрос может приводить к излишним закупкам дополнительных ресурсов и связан исключительно с возможностями конструкции системы. PowerScale позволяет обойти эту проблему.

Архитектура PowerScale автоматически оптимизирует степень расслоения памяти в зависимости от того, какие платы памяти инсталлированы в системе. В зависимости от конкретной конфигурации она будет использовать низкую или высокую степень расслоения или их комбинацию. Все это полностью прозрачно для программного обеспечения и, что более важно, для пользователя.

Архитектура матричного коммутатора

Архитектура коммутатора реализована с помощью аппаратной сети, которая осуществляет индивидуальные соединения типа точка-точка процессора с процессором, процессора с основной памятью и процессора с магистралью данных ввода/вывода. Эта сеть работает совместно с разделяемой адресной шиной. Такой сбалансированный подход позволяет использовать лучшие свойства каждого из этих методов организации соединений.

Разделяемая адресная шина упрощает реализацию наблюдения (snooping) за адресами, которое необходимо для аппаратной поддержки когерентности памяти. Адресные транзакции конвейеризованы, выполняются асинхронно (расщеплено) по отношению к пересылкам данных и требуют относительно небольшой полосы пропускания, гарантируя, что этот ресурс никогда войдет в состояние насыщения.

Организация пересылок данных требует больше внимания, поскольку уровень трафика и время занятости ресурсов физического межсоединения здесь существенно выше, чем это требуется для пересылки адресной информации. Операция пересылки адреса представляет собой одиночную пересылку, в то время как операция пересылки данных должна удовлетворять требованию многобайтной пересылки в соответствии с размером строки кэша ЦП. При реализации отдельных магистралей данных появляется ряд дополнительных возможностей, которые обеспечивают:

- • максимальную скорость передачи данных посредством соединений точка-точка на более высоких тактовых частотах;
- • параллельную пересылку данных посредством организации выделенного пути для каждого соединения;

- • разделение адресных транзакций и транзакций данных. Поэтому архитектуру PowerScale компании Bull можно назвать многопотоковой аппаратной архитектурой (multi-threaded hardware architecture) с возможностями параллельных операций.

На рис. 12.3 показаны основные режимы и операции, выполняемые матричным коммутатором.

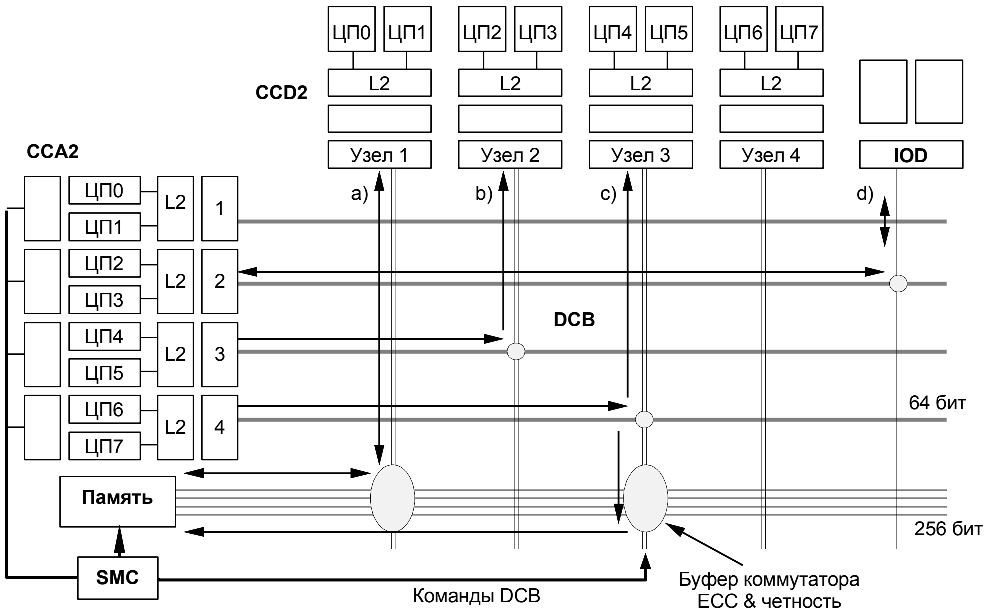


Рис. 12.3. Матричный коммутатор. CCA2 - сдвоенный контроллер адресов кэш-памяти; CCD2 - сдвоенный контроллер данных кэш-памяти; IOD - дочерняя плата ввода/вывода; DCB - матричный коммутатор данных; SMC - контроллер системной памяти

Режим обращения к памяти - Memory mode: (a)

Процессорный узел или узел в/в коммутируется с массивом памяти (МА). Такое соединение используется для организации операций чтения памяти или записи в память.

Режим вмешательства (чтение): (b)

Читающий узел коммутируется с другим узлом (вмешивающимся узлом) и шиной данных МА. Этот режим используется тогда, когда при выполнении операции чтения строки от механизма наблюдения за когерентным состоянием памяти поступает ответ, что данная строка находится в кэш-памяти другого узла и модифицирована. В этом случае данные, извлекаемые из строки кэша владельца, подаются читающему узлу и одновременно записываются в МА. Если читающий и вмешивающийся ЦП находятся внутри одного и того же узла, то данные заворачиваются назад на уровне узла и одновременно записываются в память.

Режим вмешательства (чтение с намерением модификации - RWITM):(c)

Процессорный узел или узел в/в (читающий узел) коммутируется с другим процессорным узлом или узлом в/в. Этот режим используется тогда, когда при выполнении операция RWITM от механизма наблюдения поступает ответ, что данная строка находится в кэш-памяти другого узла и модифицирована. В этом случае данные, извлекаемые из строки кэша владельца, подаются только читающему узлу и не записываются в память.

Режим программируемого ввода/вывода (PIO): (d)

Процессорный узел коммутируется с узлом в/в. Это случай операций PIO, при котором данные обмениваются только между процессором и узлом в/в.

Режим в/в с отображением в памяти (memory mapped):

Главный узел коммутируется с узлами в/в (подчиненными узлами), вовлеченными в транзакцию. Это случай операций с памятью.

Параметры производительности

Вслед за установочной фазой транзакции (например, после установки адреса на адресной шине) данные могут пересылаться через коммутатор на полной скорости синхронизации. Это возможно благодаря организации соединению точка-точка, которое создается для каждой отдельной транзакции. Поэтому в дальнейшем какие-либо помехи отсутствуют. Возможно также выполнять параллельно несколько операций, например, множественный доступ к памяти или пересылки между кэшами.

Для того чтобы уменьшить задержку памяти, операции чтения начинаются до выполнения каких-либо действий по обеспечению глобальной когерентности на уровне системы. Ответы когерентности полностью синхронизированы, разрешаются за фиксированное время и поступают всегда прежде, чем будет захвачен разделяемый ресурс - шина памяти. Это помогает избежать ненужных захватов шины. Любые транзакции, которые не разрешаются когерентно за данное фиксированное время, позднее будут повторены системой.

Используемая в системе внутренняя частота синхронизации равна 75 МГц, что позволяет оценить уровень производительности разработанной архитектуры. Интерфейс физической памяти имеет ширину 32 байта и, учитывая арбитраж шины, позволяет пересылать 32 байта каждые 3 такта синхронизации. Это дает скорость передачи данных 800 Мбайт/с, поддерживаемую на уровне интерфейса памяти. Каждый порт ЦП имеет ширину 8 байт и способен передавать по 8 байт за такт, т.е. со скоростью 600 Мбайт/с. Следует отметить, что это скорость, достигаемая как при пересылке ЦП-память, так и при пересылке кэш-кэш. Скорость 800 Мбайт/с для памяти поддерживается с помощью буферов в коммутаторе, которые позволяют конвейеризовать несколько операций.

Поскольку несколько операций могут выполняться через коммутатор на полной скорости параллельно, то для оптимальной смеси операций (две пересылки из ЦП в память, плюс пересылка кэш-кэш) пропускная способность может достигать пикового значения 1400 Мбайт/с. Таким образом, максимальная пропускная способность будет варьироваться в диапазоне от 800 до 1400 Мбайт/с в зависимости от коэффициента попаданий кэш-памяти.

Когерентность кэш-памяти

Известно, что требования, предъявляемые современными процессорами к полосе пропускания памяти, можно существенно сократить путем применения больших многоуровневых кэшей. Проблема когерентности памяти в мультипроцессорной системе возникает из-за того, что значение элемента данных, хранящееся в кэш-памяти разных процессоров, доступно этим процессорам только через их индивидуальные кэши. При этом определенные операции одного из процессоров могут влиять на достоверность данных, хранящихся в кэшах других процессоров. Поэтому в подобных системах жизненно необходим механизм обеспечения когерентного (согласованного) состояния кэшей. С этой целью в архитектуре PowerScale используется стратегия обратной записи, реализованная следующим образом.

Вертикальная когерентность кэшей

Каждый процессор для своей работы использует двухуровневый кэш со свойствами охвата. Это означает, что кроме внутреннего кэша первого уровня (кэша L1), встроенного в каждый процессор PowerPC, имеется связанный с ним кэш второго уровня (кэш L2). При этом каждая строка в кэше L1 имеется также и в кэше L2. В настоящее время объем кэша L2 составляет 1 Мбайт на каждый процессор, а в будущих реализациях предполагается его расширение до 4 Мбайт. Сама по себе кэш-память второго уровня позволяет существенно уменьшить число обращений к памяти и увеличить степень локализации данных. Для повышения быстродействия кэш L2 построен на принципах прямого отображения. Длина строки равна 32 байт (размеру когерентной гранулированности системы). Следует отметить, что, хотя с точки зрения физической реализации процессора PowerPC, 32 байта составляют только половину строки кэша L1, это не меняет протокол когерентности, который управляет операциями кэша L1 и гарантирует что кэш L2 всегда содержит данные кэша L1.

Кэш L2 имеет внешний набор тегов. Таким образом, любая активность механизма наблюдения за когерентным состоянием кэш-памяти может быть связана с кэшем второго уровня, в то время как большинство обращений со стороны процессора могут обрабатываться первичным кэшем. Если механизм наблюдения обнаруживает попадание в кэш второго уровня, то он должен выполнить арбитраж за первичный кэш, чтобы обновить состояние и возможно найти данные, что обычно будет приводить к приостановке процессора. Поэтому глобальная память может работать на уровне тегов кэша L2, что позволяет существенно ограничить количество операций наблюдения, генерируемых системой в направлении данного процессора. Это, в свою очередь, существенно увеличивает производительность системы, поскольку любая операция наблюдения в направлении процессора сама по себе может приводить к приостановке его работы.

Вторичная когерентность кэш-памяти

Вторичная когерентность кэш-памяти требуется для поддержки когерентности кэшей L1&L2 различных процессорных узлов, т.е. для обеспечения когерентного состояния всех имеющихся в мультипроцессорной системе распределенных кэшей (естественно включая поддержку когерентной буферизации ввода/вывода как по чтению, так и по записи).

Вторичная когерентность обеспечивается с помощью проверки каждой транзакции, возникающей на шине MPB_SysBus. Такая проверка позволяет обнаружить, что запрашиваемая по шине строка уже кэширована в процессорном узле, и обеспечивает выполнение необходимых операций. Это делается с помощью тегов кэша L2 и логически поддерживается тем фактом, что L1 является подмножеством L2.

Протокол MESI и функция вмешательства

В рамках архитектуры PowerScale используется протокол MESI, который представляет собой стандартный способ реализации вторичной когерентности кэш-памяти. Одной из основных задач протокола MESI является откладывание на максимально возможный срок операции обратной записи кэшированных данных в глобальную память системы. Это позволяет улучшить производительность системы за счет минимизации ненужного трафика данных между кэшами и основной памятью. Протокол MESI определяет четыре состояния, в которых может находиться каждая строка каждого кэша системы. Эта информация используется для определения соответствующих последующих операций (рис. 12.4).

Состояние строки "Единственная" (Exclusive):

Данные этой строки достоверны в данном кэше и недостоверны в любом другом кэше. Данные не модифицированы по отношению к памяти.

Состояние строки "Разделяемая" (Shared):

Данные этой строки достоверны в данном кэше, а также в одном или нескольких удаленных кэшах.

Состояние строки "Модифицированная" (Modified):

Данные этой строки достоверны только в данном кэше и были модифицированы. Данные недостоверны в памяти.

Состояние строки "Недостоверная" (Invalid):

Достоверные данные не были найдены в данном кэше.

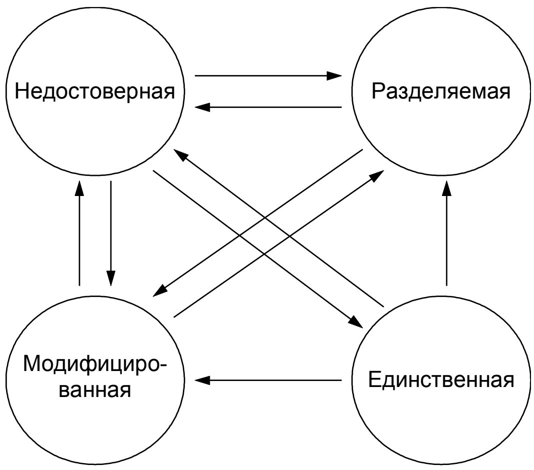


Рис. 12.4. Диаграмм переходов состояний протокола MESI

Для поддержки мультипроцессорной организации были реализованы несколько примитивов адресной шины. Это позволяет одному главному устройству шины передавать, а другим устройствам обнаруживать (или наблюдать) появление этих примитивов на шине. Устройство-владелец кэша наблюдает за адресной шиной во время глобального запроса и сравнивает целевой адрес с адресами тегов в своем кэше L2. Если происходит попадание, то выполняемые действия определяются природой запроса.
Как уже было отмечено, одной из функций тегов L2 является уменьшение накладных расходов, связанных с ответами на запросы механизма наблюдения. Доступ к тегам L2 разделяется между процессорами и адресной шиной. Теги L2 практически выполняют роль фильтров по отношению к активностям наблюдения. Это позволяет процессорам продолжать обработку вместо того, чтобы отвечать на каждый запрос наблюдения. Хотя теги L2 представляют собой разделяемый между процессором и шиной ресурс, его захват настолько кратковременен, что практически не приводит ни к каким конфликтам.
Состояние строки кэш-памяти "модифицированная" означает в частности то, что кэш, хранящий такие данные, несет ответственность за правильность этих данных перед системой в целом. Поскольку в основной памяти эти данные недостоверны, это означает, что владелец такого кэша должен каким-либо способом гарантировать, что никакой другой модуль системы не прочитает эти недостоверные данные. Обычно для описания такой ответственности используется термин "вмешательство" (intervention), которое представляет собой действие, выполняемое устройством-владельцем модифицированных кэшированных данных при обнаружении запроса наблюдения за этими данными. Вмешательство сигнализируется с помощью ответа состоянием "строка модифицирована" протокола MESI, за которым следуют пересылаемые запросчику, а также потенциально в память, данные.
Для увеличения пропускной способности системы в PowerScale реализованы два способа выполнения функции вмешательства:

- • Немедленная кроссировка (Cross Immediate), которая используется когда канал данных источника и получателя свободны? и можно пересылать данные через коммутатор на полной скорости.
- • Поздняя кроссировка (Cross Late), когда ресурс (магистраль данных) занят, поэтому данные будут записываться в буфер коммутатора и позднее пересылаться запросчику.

Физическая реализация архитектуры
Ниже на рис. 12.5 показана схема, представляющая системные платы, разработанные компанией Bull, которые используются для физической реализации архитектуры PowerScale.
Многопроцессорная плата:
Многопроцессорная материнская плата, которая используется также в качестве монтажной панели для установки модулей ЦП, модулей основной памяти и одной платы в/в (IOD).
Модуль ЦП (дочерняя процессорная плата):
Каждый модуль ЦП, построенный на базе PowerPC 601/604, включает два микропроцессора и связанные с ними кэши. Имеется возможность модернизации системы, построенной на базе процессоров 601, путем установки модулей ЦП с процессорами 604. Смешанные конфигурации 601/604 не поддерживаются.
Дочерняя плата ввода/вывода: (IOD)
IOD работает в качестве моста между шинами MCA и комплексом ЦП и памяти. Поддерживаются 2 канала MCA со скоростью передачи 160 Мбайт/с каждый. Хотя поставляемая сегодня подсистема в/в базируется на технологии MCA, это не является принципиальным элементом архитектуры PowerScale. В настоящее время проводятся исследования возможностей реализации нескольких альтернативных шин ввода/вывода, например, PCI.

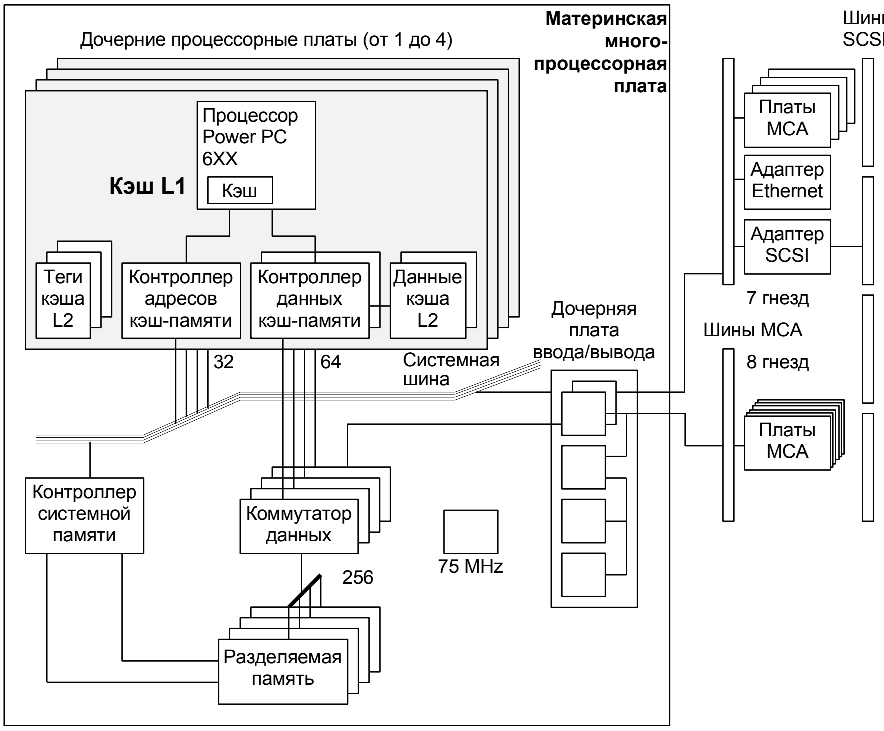


Рис. 12.5. Физическая реализация PowerScale

Платы памяти:

Каждая плата памяти состоит из четного числа банков. Максимальное число банков равно 16. Объем памяти на каждой плате может быть 64, 256 или 512 Мбайт.

Коммутатор данных (DCB) интегрирован в нескольких СБИС (4х16 бит) и функционально соединяет магистраль данных MPB_SysBus с подсистемой памяти, модулями ЦП и платой в/в. Ширина магистрали данных DCB на уровне массива памяти составляет 256 + 32 бит, а ширина магистрали данных для каждого порта ЦП и порта в/в равна 64 + 8 бит. Операции DCB управляются контроллером системной памяти (SMC) с помощью командной шины, обеспечивающей необходимую коммутацию устройств.

8.1.4. **12.1.4. Семейство UNIX-серверов Escala**

На российский рынок в настоящее время активно продвигаются UNIX-серверы семейства Escala - многопроцессорные системы с архитектурой PowerScale, построенные на базе микропроцессора PowerPC 601. Они предлагаются для работы в качестве нескольких типов серверов приложений: сервера транзакций (Transation Server) при использовании мониторов обработки транзакций подобных Tuxedo, сервера базы данных (Database Server) на основе таких известных систем как Oracle или Informix, а также управляющего сервера (System Management Server), обеспечивающего управление инфраструктурой предприятия и существенно упрощающего администрирование гетерогенными системами и сетями. Серверы Escala двоично совместимы с системами Bull DPX/20, а их архитектура разработана с учетом возможности применения новейших процессоров PowerPC 604 и 620.

Таблица 12.1

	M101	**M201**
	Mini-Tower	
	PowerPC 601	

	M101	M201
	Mini-Tower	
	75	75
	1/4	2/4
	512	512
	32	64
	512	512
	MCA	MCA
	160	160
	6	6
	4	4
	2	2
	1/18	1/18
	738	738
	1/4	1/4
	77	77
	84	84
	3600/6789	3600/6789
	3900/7520	3900/7520
	750/1350	750/1350

Основные характеристики серверов Escala в зависимости от применяемого конструктива даны в таблице 12.1. Системы семейства Escala обеспечивают подключение следующих коммуникационных адаптеров: 8-, 16- и 128-входовых адаптеров асинхронных последовательных портов, 1- или 4-входовых адаптеров портов 2 Мбит/с X.25, а также адаптеров Token-Ring, Ethernet и FDDI.

Заключение

При разработке PowerScale Bull удалось создать архитектуру с достаточно высокими характеристиками производительности в расчете на удовлетворение нужд приложений пользователей сегодня и завтра. И обеспечивается это не только увеличением числа процессоров в системе, но и возможностью модернизации поколений процессоров PowerPC. Разработанная архитектура привлекательна как для разработчиков систем, так и пользователей, поскольку в ее основе лежит одна из главных процессорных архитектур в компьютерной промышленности - PowerPC.

8.2. 12.2. Серверы компании DEC

Корпорация Digital Equipment (далее по тексту - Digital) широко известна в мире и является одной из крупнейших компьютерных компаний, компьютеры которой остаются популярными уже в течение почти 40 лет (начиная с ее основания в 1957 году и выпуска первых машин PDP-1 в 1960 г.).

Продукция Digital чрезвычайно разнообразна и включает в себя следующие основные направления: компьютерную технику - от персональных компьютеров до мощных информационных центров; телекоммуникационные устройства для локальных и распределенных сетей и для всех видов передающей физической среды - от коннекторов, осуществляющих физическое соединение различных устройств, до мощных высокоскоростных модульных концентраторов, включающих в себя маршрутизаторы, мосты, шлюзы, повторители и др.; периферийные устройства, принтеры, мультимедиа; программное обеспечение, включающее операционные системы, сетевые программные средства, системы электронной почты и электронного обмена данными, распределенную офисную среду, средства разработки приложений и др.

Компания Digital широко известна своими сериями мини-ЭВМ PDP-11 и VAX, работающими под управлением операционных систем RSX11M и VMS соответственно. Имея огромную базу установленных по всему миру систем этого типа, компания Digital вынуждена ее поддерживать и в настоящее время выпускает целый ряд совместимых с PDP и VAX изделий. Однако, понимая необходимость перехода на стандарты открытых систем, Digital разработала новую стратегию, ключевым камнем которой является новая технология, основанная на 64-битовой суперскалярной архитектуре Alpha.

В настоящее время корпорация Digital сконцентрировала основные усилия на разработке и производстве современных 64-разрядных RISC-систем. Новейший микропроцессор Alpha DECchip 21164 на сегодня является самым быстрым микропроцессором. Архитектура Alpha полностью сохраняет преемственность поколений компьютеров: практически все программное обеспечение ЭВМ VAX работает и на новых системах Alpha.

Кроме того, в течение последних лет корпорация Digital применяет свой огромный опыт для производства персональных компьютеров, которые завоевывают все более прочное положение на современном рынке.

8.2.1. 12.2.1. Семейство компьютеров Alpha

Отличительная черта платформы Alpha - это сбалансированность. Благодаря 64-разрядной архитектуре и высокоскоростным каналам связи с периферией Alpha поддерживает работу с огромными массивами данных, как на дисках, так и в оперативной памяти, что является весьма критичным для многих приложений.

Другим отличительным качеством платформы Alpha является ее универсальность с точки зрения применения различных операционных систем. Это и всем известные NetWare и Pick, а также DECelx - операционная система реального времени. OpenVMS, операционная система компьютеров VAX, продолжает свой путь на платформе Alpha. Digital UNIX - UNIX нового поколения, унифицированный, удовлетворяющий стандартам Open Software Foundation (OSF) и обеспечивающий совместимость с BSD и System V, - также функционирует на платформе Alpha, которая поддерживает и операционную систему Windows NT - продукт Microsoft.

Названия компьютеров Alpha

Названия новых моделей компьютеров семейства Alpha строятся следующим образом: AlphaServer (AlphaStation) поколение процессора/частота, где AlphaServer или AlphaStation обозначает, соответственно, сервер или рабочую станцию, поколение процессоров представляется цифрой 4 для процессоров 2106* и цифрой 5 - для 21164.

8.2.2. 12.2.2. Серверы на базе Alpha

Семейство серверов Alpha представляет собой полный ряд систем: от минимальной конструкции до сервера крупной распределенной сети. Ниже дано описание основных свойств этих компьютеров и средств их реализации.

Высокая надежность и доступность:

- • "Горячее" переключение дисков, т.е. внутренний диск может быть заменен во время работы сервера.
- • Код коррекции ошибок (ECC, Error Correcting Code). Серверы Alpha включают ECC для основной и кэш- памяти. При использовании этой технологии происходит постоянная проверка памяти, причем при этом ошибки не только обнаруживаются, но и автоматически корректируются.
- • Технология дублирования дисков (Redundant Array of Inexpensive Disks, RAID)
- • Двойная шина SCSI.
- • Дублирование источников питания.

- • Автоматический перезапуск системы. При сбое в операционной системе эта возможность минимизирует время недоступности системы.
- • Управление температурным режимом. Системы AlphaServer включают температурные и другие датчики, позволяющие следить за состоянием системы.

Открытая архитектура:

- • Шина PCI, обеспечивающая скорость передачи 132 Мб/с и соответствие международным стандартам.
- • Стандартные слоты EISA, предоставляющие возможность использования большого количества стандартных карт.
- • Высокоскоростной интерфейс SCSI-2 для подключения до 7 периферийных устройств, обеспечивающие в два раза более высокую скорость передачи шины SCSI и возможность подключения различных стандартных периферийных устройств.
- • Сетевые опции, включающие Ethernet, Token Ring, FDDI.

Средства управления:

- • Реализация удаленного управления.
- • Расширенные средства диагностики.
- • Получение информации о конфигурации системы.
- • Программное обеспечение управления нестандартными ситуациями и журналы диагностики сбоев.

Расширяемость/наращиваемость:

- • Возможность обновления процессора ("upgrade").
- • Возможность подключения внешней памяти.
- • Использование симметричной мультипроцессорной обработки (Symmetric Multi-Processing, SMP), позволяющей добавлять дополнительные процессоры.
- • Гибкость выбора операционной системы (OpenVMS AXP, Digital UNIX, Microsoft Windows NT).

Использование кластеров:

- • Возможность построения кластерных систем.

Основные характеристики серверов AlphaServer представлены в таблице 12.2.

Таблица 12.2.

Система/ Характеристики	**AlphaServer 400**	**AlphaServer 1000**	**AlphaServer 2000**	**AlphaServer 2100**	**AlphaServer 8200**	**AlphaServer 8400**
Частота	166 МГц	233 МГц	4/275:275 МГц	5/250:250 МГц	300 Мгц 4/233:233 МГц	300 МГц 4/275:275 МГц 4/200:200 МГц
Число процессоров	1	1	1-2	1-4	1-6	1-12
Число транзакций в секунду (TpsA)	До 130	До 300	4/275:До 625 4/233:До 400	5/250:До 1200 4/275:До 850 4/200:До 675	Свыше 2000	Свыше 3000

Максимальная память	192 Мб	512 Мб	4/275: 1 Гб 4/233:640 Мб	2 Гб	6 Гб	14 Гб
Память на диске	441 Гб	220 Гб	Свыше 500Гб	500 Гб	10 Тб	10 Тб
Поддержка ввода/вывода	2 слота PCI 3 слота EISA 1 слот PCI/EISA	2 слота PCI 7 слотов EISA	3 слота PCI 7 слотов EISA	3 слота PCI 8 слотов EISA	108 слотов PCI 8 слотов EISA	144 слота PCI 8 слотов EISA 1 слот PCI/EISA
ECC память	Нет	Да	Да	Да	Да	Да
RAID	Да	Да	Да	Да	Да	Да
Авто перезагрузка	Да	Да	Да	Да	Да	Да
Дублирование питания	Нет	Да	Да	Да	Да	Да
Управление температурой	Нет	Да	Да	Да	Да	Да

8.2.2.1. *12.2.2.1. AlphaServer 8400*

AlphaServer 8400 - это реализация сервера на базе микропроцессора DECchip 21164 (частота - от 300 МГц) высокопроизводительного сервера масштаба предприятия. AlphaServer 8400 поддерживает до 12 процессоров, 14 Гб памяти и скорость ввода/вывода свыше 1,2 Гб/сек. Сбалансированная конструкция и быстрые процессоры позволяют обеспечивать обработку более 3000 транзакций в секунду. На AlphaServer 8400 могут функционировать операционные системы OpenVMS, Digital UNIX и Windows NT. Для защиты инвестиций своих пользователей Digital предлагает линию модернизации таких продуктов, как DEC 7000 и VAX 7000. Этот upgrade до уровня AlphaServer 8400 может быть выполнен в течение нескольких часов, что обеспечивает высокую производительность сегодня и средства роста в будущем. Архитектура AlphaServer 8400 разработана с учетом возможности использования будущих поколений микропроцессора Alpha. AlphaServer 8400 оснащается высокоскоростными шинами ввода/вывода PCI (144 слота на 12 физически различных шинах). Данный компьютер имеет относительно низкую стоимость в своем классе и может использоваться в качестве сервера крупной распределенной базы данных, обеспечивая при этом надежность и готовность на уровне более дорогих мэйнфреймов.

8.2.2.2. *12.2.2.2. AlphaServer 8200*

Компьютер AlphaServer 8200 - это одна из наиболее высокопроизводительных систем для офиса в современной промышленности. Его конфигурация может включать до шести микропроцессоров DECchip 21164. Имея все преимущества 64-разрядной Alpha-архитектуры, до 6 Гб памяти и до 108 слотов PCI, данный сервер обеспечивает возможности роста даже для самых крупных и сложных приложений. AlphaServer 8200 поддерживает операционные системы OpenVMS, Digital UNIX и Windows NT. Небольшие предприятия и крупные подразделения могут использовать производительность, мощность и надежность этого сервера для приложений, которые прежде функционировали на системах масштаба крупного предприятия. Большие базы данных, процессы моделирования, системы поддержки принятия решений - вот несколько примеров приложений, которые легко поддерживаются AlphaServer 8200.

8.2.2.3. *12.2.2.3. AlphaServer 2100*

Системы AlphaServer 2100 представляют собой недорогие SMP-серверы, базирующиеся на шинах PCI/EISA. Они поддерживают операционные системы OpenVMS, Digital UNIX и Windows NT. Данные компьютеры могут использоваться в качестве серверов высокопроизводительных коммерческих приложений и баз данных, а также серверов крупных локальных сетей. AlphaServer 2100 4/233 (микропроцессор DECchip 21064A) имеет частоту 233 МГц с кэш-памятью 1 Мб; AlphaServer 2100 4/275 (микропроцессор DECchip 21064A) - 275 МГц с кэш-памятью 4 Мб;

AlphaServer 2100 5/250 (микропроцессор DECchip 21164) - 250 МГц с кэш-памятью 4 Мб. Каждая система может иметь конфигурацию с 1-4 процессорами, поддерживает до 2 Гб оперативной памяти и до 64 Гб внутренней дисковой памяти. Пропускная способность системной шины равна 667 Мб/сек, а высокопроизводительная подсистема ввода/вывода PCI имеет пиковую пропускную способность 132 Мб/сек. Шина ввода/вывода EISA (33 Мб/сек) поддерживает широкий спектр стандартных устройств.

8.2.2.4. *12.2.2.4. AlphaServer 2000*

Системы AlphaServer 2000 близки по своему уровню к компьютерам AlphaServer 2100. Они также представляют собой недорогие SMP-серверы, базирующиеся на шинах PCI/EISA, и поддерживают операционные системы OpenVMS, Digital UNIX и Windows NT. Данные компьютеры могут использоваться в качестве серверов высокопроизводительных коммерческих приложений и баз данных, а также серверов крупных локальных сетей. AlphaServer 2000 4/233 (микропроцессор DECchip 21064A) имеет частоту 233 МГц с кэш-памятью 1 Мб; AlphaServer 2000 4/275 (микропроцессор DECchip 21064A) - 275 МГц с кэш-памятью 4 Мб. Каждая система может иметь конфигурацию с 1-2 процессорами, поддерживает до 1 Гб оперативной памяти и до 32 Гб внутренней дисковой памяти. Пропускная способность системной шины равна 667 Мб/сек, а высокопроизводительная подсистема ввода/вывода PCI имеет пиковую пропускную способность 132 Мб/сек. Шина ввода/вывода EISA (33 Мб/сек) поддерживает широкий спектр стандартных устройств.

8.2.2.5. *12.2.2.5. AlphaServer 1000*

Недорогой сервер AlphaServer 1000 4/233 относится к уровню серверов рабочих групп. Данный компьютер может быть поставлен с заранее установленным программным обеспечением доступа к сети Internet. AlphaServer 1000, имеющий 512 Мб памяти ECC, более 14 Гб внутренней памяти "горячего" переключения, шину PCI/EISA, две шины PCI и 7 портов расширения EISA, дисковод для гибкого диска, CD-ROM. Следует также учитывать возможность выбора одной их трех операционных систем: Windows NT, Digital UNIX и OpenVMS.

8.2.2.6. *12.2.2.6. AlphaServer 400*

AlphaServer 400 4/166- это небольшой сервер, замыкающий снизу семейство машин AlphaServer. AlphaServer 400 поддерживают три операционные системы: Digital UNIX, Windows NT и OpenVMS. Система включает в себя стандартную шину PCI, обеспечивающую широкий выбор периферийных устройств. Компьютер AlphaServer 400 4/166 призван заменить на рынке такой популярный продукт Digital, как миникомпьютер MicroVAX 3100. Имея те же преимущества, что и MicroVAX, современная система AlphaServer 400 имеет и дополнительные достоинства: открытость архитектуры, использование стандартных компонент (например, шина PCI) и, главное, мощный микропроцессор Alpha. AlphaServer 400 имеет многие преимущества больших систем, такие, как высокая надежность и расширенные средства обеспечения безопасности и управления. Этот сервер работает на частоте 166 МГц и является одним из самым "быстрых" серверов в своем классе. Это подтверждают результаты тестов: AlphaServer 400 4/166 выполняет 100 транзакций в секунду по тесту TPC-A и имеет показатели 107.7 и 134.8 единиц, соответственно, на тестах SPECint92 и SPECfp92.

AlphaServer 400 может служить в качестве сервера локальной сети небольших и средних по размеру организаций. Следует отметить, что Digital предлагает недорогие способы увеличения производительности этих систем путем замены процессоров на самые современные модели по мере их появления. AlphaServer 400 имеет до 192 Мб оперативной памяти, может использовать более 8 Гб внутренней и до 44 Гб - общей дисковой памяти. Особенность этого компьютера - средства удаленной диагностики, позволяющие выполнять программы тестирования и конфигурации системы удаленным способом по телефонным линиям связи.

8.3. **12.3. Серверы компании Hewlett-Packard**

Компания Hewlett-Packard была учреждена в Калифорнии в 1938 году с целью создания электронного тестирующего и измерительного оборудования. В настоящее время компания разрабатывает, производит, осуществляет маркетинг и сервис систем для коммерческих приложений, автоматизации

производственных процессов, процессов разработки, тестирования и измерений, а также аналитические и медицинские инструменты и системы, периферийное оборудование, калькуляторы и компоненты для использования в широком ряде отраслей промышленности. Она продает более 4500 изделий, используемых в промышленности, бизнесе, науке, образовании, медицине и инженерии.
Основой разработки современных компьютеров Hewlett-Packard является архитектура PA-RISC. Она была разработана компанией в 1986 году, и с тех пор, благодаря успехам интегральной технологии, прошла несколько стадий своего развития от многокристального до однокристального исполнения. Архитектура PA-RISC разрабатывалась с учетом возможности построения многопроцессорных систем, которые реализованы в старших моделях серверов.

8.3.1. **12.3.1. Серверы HP9000 класса D**

В секторе рынка серверов рабочих групп компания HP представлена довольно широкой серией систем HP9000 класса D. Это серия систем с относительно низкой стоимостью, которая конкурирует с серверами, построенными на базе ПК. Эти системы базируются на архитектуре процессоров PA-RISC (75 и 100 МГц PA-7100LC, 100 и 120 МГц PA-7200, а также 160 МГц PA-8000) и работают под управлением операционной системы HP-UX.
Модели D200, D210 и D310 представляют собой однопроцессорные системы. Модели D250, D260, D270 и D350 могут оснащаться как одним, так и двумя процессорами. В своих моделях D3XX HP подчеркивает свойства обеспечения высокой готовности: возможность "горячей" замены внутренних дисковых накопителей, возможность организации дискового массива RAID и наличие источника бесперебойного питания. Эти модели обладают также расширенными возможностями по наращиванию оперативной памяти и подсистемы ввода/вывода.
В моделях D2XX имеется 5 гнезд расширения ввода/вывода и 2 отсека для установки дисковых накопителей с интерфейсом SCSI-2. В моделях D3XX количество гнезд расширения ввода/вывода расширено до 8, в 5 отсеках могут устанавливаться дисковые накопители с интерфейсом Fast/Wide SCSI-2, которые допускают замену без выключения питания системы.
Старшие модели серии обеспечивают возможность расширения оперативной ECC-памяти до 1.5 Гбайт, при этом коэффициент расслоения может увеличиваться до 12. Максимальный объем дискового пространства при использовании внешних дисковых массивов может достигать 5.0 Тбайт.

Таблица 12.3. Основные характеристики серверов HP 9000 класса D

МОДЕЛЬ	**D200/ D210**
ЦП	
Тип процессора	PA7100LC
Тактовая частота (МГц)	75(D200) 100(D210)
Число процессоров	1
Пропускная способность системной шины (Мб/сек)	960
Размер кэша (Кб) (команд/данных)	256
ПАМЯТЬ	
Минимальный объем (Мб)	32
Максимальный объем (Гб)	0.5
ВВОД/ВЫВОД	
Тип шины	HP-HSC
Количество слотов	5 1EISA 1HSC 3EISA/ HSC Comb
Максимальная пропускная способность подсистемы в/в (Мб/сек)	160

МОДЕЛЬ	D200/ D210
Количество отсеков для дисков Fast/Wide SCSI-2	2(Opt)
Количество отсеков для дисков SCSI-2	2
Количество отсеков для НМЛ и CD-ROM	3
Максимальная емкость дисковой памяти (Тб)	0.5
Количество послед. портов	2
Количество парал. портов	1
Сетевые интерфейсы	Ethernet

Пиковая пропускная способность системной шины составляет 960 Мбайт/с. Все модели оснащаются высокоскоростными шинами в/в HP-HSC (high speed connect) с пропускной способностью 160 Мбайт/с. Установка двух HP-HSC в моделях D3XX обеспечивает удвоение пропускной способности ввода/вывода этих систем (до 320 Мбайт/с). В таблицах 12.3 и 12.4 приведены основные характеристики серверов класса D.

Таблица 12.4. Производительность серверов HP9000 класса D

	1 ЦП PA7100LC 75 Мгц D200	**1 ЦП PA7100LC 100 Мгц D210/310**	**1 ЦП PA7200 100 Мгц D250/350**	**1 ЦП PA8000 160 Мгц D270/370**	**2 ЦП PA7200 100 Мгц D250/350**	**2 ЦП PA7200 120 Мгц D260/360**	**2 ЦП PA8000 160 Мгц D270/370**
SPECint95	2.2	2.9	3.6	10.4	-	-	-
SPECfp95	2.9	3.7	6.1	17.9	-	-	-
SPECrate_int95	19.2	26.5	32.5	90	64.5	81.4	180
SPECrate_fp95	25.8	33.3	55.1	150	97.4	131	240
tpmC	-	-	-	-	-	-	5822

8.3.2. **12.3.2. Серверы HP9000 класса K**

Серверы HP9000 класса K представляют собой системы среднего класса, поддерживающие симметричную мультипроцессорную обработку (до 4 процессоров). Также как и системы класса D они базируются на архитектуре PA-RISC (120 МГц PA-7200 с кэш-памятью команд/данных первого уровня 256/256 Кбайт или 1/1 Мбайт, а также 160 и 180 МГц PA-8000 с кэш-памятью команд/данных первого уровня 1/1 Мбайт, работающей на тактовой частоте процессора). Конструкция серверов класса K обеспечивает высокую пропускную способность систем. Основными компонентами поддержания высокой производительности являются системная шина с пиковой пропускной способностью 960 Мбайт/с, большая оперативная память с контролем и исправлением одиночных ошибок (ECC) емкостью до 4 Гбайт с 32-кратным расслоением, многоканальная подсистема ввода/вывода с пропускной способностью до 288 Мбайт/с, стандартная высокоскоростная шина Fast/Wide Differential SCSI-2, а также дополнительные возможности по подключению высокоскоростных сетей и каналов типа FDDI, ATM и Fibre Channel. В конструкции сервера предусмотрены 4 отсека для установки дисковых накопителей, а с помощью специальных стоек расширения емкость дисковой памяти системы может быть доведена до 8.3 Тбайт. Основные параметры серверов HP9000 класса K представлены в таблицах 12.5 и 12.6.

Таблица 12.5. Производительность серверов HP9000

	1 ЦП K210 K410	1 ЦП K220 K420
SPECint95	4.4	4.6
SPECfp95	7.4	8.2
SPECrate_int95	39.7	41.3

SPECrate_fp95	66.6	73.2

	2 ЦП K210 K410	
SPECrate_int95	77.2	
SPECrate_fp95	127.0	
	4 ЦП K210 K410	
SPECrate_int95	152.0	
SPECrate_fp95	217.0	
tpmC	-	
LADDIS (IOPS)	-	

Таблица 12.6. Основные характеристики серверов HP 9000 класса K

МОДЕЛЬ	**K210**	**K22**
ЦП		
Тип процессора	PA7200	PA72
Тактовая частота (МГц)	120 100(D210)	120
Число процессоров	1-4	1-4
Пропускная способность системной шины (Мб/сек)	960	960
Размер кэша (Кб) (команд/данных)	256/ 256	1024 1024
ПАМЯТЬ		
Минимальный объем (Мб)	64	128
Максимальный объем (Гб)	2.0	2.0
ВВОД/ВЫВОД		
Количество слотов HP-HSC	1	1
Количество слотов HP-PB	4	4
Максимальная пропускная способность подсистемы в/в (Мб/сек)	288	288
Количество отсеков для дисков Fast/Wide SCSI-2	4	4
Максимальная емкость дисковой памяти (Тб)	3.8	3.8
Количество последовательных портов	2	2
Количество параллельных портов	1	1
Сетевые интерфейсы	Ethernet	Ether

8.3.3. **12.3.3. Симметричные многопроцессорные серверы HP9000 класса T**

Самым мощным и расширяемым рядом корпоративных серверов компании HP на базе ОС UNIX является семейство HP9000 класса T. Это следующее поколение серверов, которое было разработано компанией вслед за HP9000 model 870. В начале на рынке появились системы HP9000 T500, допускающие установку до 12 процессоров PA7100, затем HP объявила 14-процессорные системы

T520, построенные на базе процессора 120 МГц PA7150. В настоящее время объявлены 12-процессорные системы T600 на базе процессора PA-8000, поставки которых должны начаться в 1997 году. Существующие системы (T500 и T520) допускают замену старых процессоров на процессоры PA-8000.

Характерной особенностью архитектуры серверов класса Т является большая емкость кэш-памяти команд (1 Мбайт) и данных (1 Мбайт) у каждого процессора системы. Серверы класса Т используют 64-битовую шину с расщеплением транзакций, которая поддерживает до 14 процессоров, работающих на частоте 120 МГц. Эффективность этой шины, как и шины Runway, составляет 80%, что обеспечивает в установившемся режиме пропускную способность 768 Мбайт/с при пиковой производительности 960 Мбайт/с.

Серверы класса Т могут поддерживать до 8 каналов HP-PB (HP Precision Bus), работающих со скоростью 32 Мбайт/с, однако в стойке основной системы поддерживается только один канал HP-PB. Для обеспечения полной конфигурации подсистемы ввода/вывода необходима установка 7 стоек расширения, занимающих достаточно большую площадь. Общая пиковая полоса пропускания подсистемы в/в в полностью сконфигурированной 8-стоечной системе составляет 256 Мбайт/с, что меньше полосы пропускания подсистемы в/в серверов класса К. Однако максимальная емкость дисковой памяти при использовании RAID-массивов достигает 20 Тбайт.

Указанная двухярусная шинная структура сервера обеспечивает оптимальный баланс между требованиями процессоров и подсистемы ввода/вывода, гарантируя высокую пропускную способность системы даже при тяжелой рабочей нагрузке. Доступ процессоров к основной памяти осуществляется посредством мощной системной шины процессор-память, поддерживающей когерентное состояние кэш-памятей всей системы. В будущих системах планируется 4-кратное увеличение пропускной способности подсистемы ввода/вывода.

Для защиты от отказов питания может применяться устанавливаемая в стойку система бесперебойного питания HP PowerTrust. Основные характеристики серверов класса Т приведены в таблице 12.7.

Таблица 12.7. Основные характеристики серверов HP 9000 класса Т

МОДЕЛЬ	**T500**
ЦП	
Тип процессора	PA7100
Тактовая частота (МГц)	90
Число процессоров	1-12
Пропускная способность системной шины (Мб/сек)	960
Размер кэша (Кб) (команд/данных)	1024/102
ПРОИЗВОДИТЕЛЬНОСТЬ	
SPECint92	135.7
SPECfp92	221.2
SPECrate_int92	3286-23717
SPECrate_fp92	5232-38780
tpmC	-
ПАМЯТЬ	
Минимальный объем (Мб)	256
Максимальный объем (Гб)	3.75
ВВОД/ВЫВОД	
Количество слотов HP-HSC	1
Количество слотов HP-PB	4
Макс. пропускная способность подсист.в/в (Мб/сек)	288

МОДЕЛЬ	T500
Максимальная емкость дисковой памяти (Тб)	20
Количество последовательных портов	16
Сетевые интерфейсы	Ethernet

8.3.4. 12.3.4. Семейство корпоративных параллельных серверов HP9000

Одним из последних продуктов, выпущенных компанией HP, является семейство параллельных систем, представленных в настоящее время двумя моделями ESP21 и ESP30. Основная концепция, лежащая в основе этих систем достаточно проста. Она заключается в создании комбинированной структуры, в которой объединяются возможности и сильные стороны проверенной временем симметричной высокопроизводительной мультипроцессорной обработки с практически неограниченным потенциалом по росту производительности и масштабируемости, который может быть достигнут посредством параллельной архитектуры. Результатом такого объединения является высокопроизводительная архитектура, обеспечивающая чрезвычайно высокую степень распараллеливания вычислений.

В отличие от некоторых других параллельных архитектур, которые используют слабо связанные однопроцессорные узлы, параллельная архитектура серверов ESP21 и ESP30 использует высокопроизводительную SMP-технологию в качестве масштабируемых строительных блоков. Преимущество такого подхода заключается в том, что прикладные системы могут пользоваться вычислительной мощностью и возможностями множества тесно связанных процессоров в инфраструктуре SMP и достаточно эффективно обеспечивать максимально возможную производительность приложений. По мере необходимости дополнительные SMP-модули могут быть добавлены в систему для увеличения степени параллелизма для масштабирования общей производительности системы, ее емкости, пропускной способности в/в, или таких системных ресурсов как основная и дисковая память.

Изделия этой серии предназначены главным образом для обеспечения масштабируемости, превышающей обычные возможности SMP-архитектуры, для крупномасштабных систем принятия решений, систем оперативной обработки транзакций, построения хранилищ данных во Всемирной Паутине Internet. Для большинства приложений модели ESP обеспечивают практически линейный рост уровня производительности. Это достигается посредством использования высокопроизводительной шинной архитектуры SMP узлов ESP в сочетании с возможностями установки дополнительных SMP-узлов с помощью разработанного компанией HP коммутатора оптоволоконных каналов (Fiber Channel Enterprise Switch). Управление всеми ресурсами системы осуществляется с единой консоли управления.

При необходимости обеспечения высокой готовности системы ESP поддерживают специальный слой программных средств MC/ServiceGuard. Эти средства позволяют создать эффективное сочетание свойств высокой производительности, масштабируемости и высокой готовности, и помимо стандартных возможностей RAS (надежности, готовности и удобства обслуживания) обеспечивают замену узлов без останова работы системы.

По сути серия EPS предоставляет средства для объединения моделей класса К (EPS21) и T(EPS30) в единую систему. 16-канальный коммутатор Fiber Channel позволяет объединить до 64 процессоров в модели EPS21 (до 256 процессоров в будущем) и до 224 процессоров в модели EPS30 (до 768 процессоров в будущем). Общая пиковая пропускная способность систем может достигать уровня 15 Гбайт/с.

Таблица 12.8. Суммарные характеристики системы при использовании в качестве коммутатора Fibre Channel Enterprise Switch модели 266

	EPS21	EPS30
Максимальное число узлов	16	16
Общее количество ЦП	64	224
Пиковая пропускная способность системной шины	15.68 Гб/с	15.68 Гб/с
Средняя пропускная способность системной шины	12.28 Гб/с	12.28 Гб/с

Пиковая пропускная способность ввода/вывода	4.4 Гб/с	4.1 Гб/с
Максимальный объем распределенной памяти	64 Гб	64 Гб
Максимальный объем распределенной дисковой памяти	132 Тб	320 Тб

Таблица 12.9. Характеристики отдельного узла EPS

	EPS21	**EPS21**	**EPS30**
Тип процессора	PA8000	PA7200	PA7200
Количество ЦП	4	4	14
Тактовая частота	180	120	120
Кэш-память команд/данных (Мб)	1/1	1/1	1/1
Стандартный объем памяти (Мб)	128	128	256
Максимальный объем памяти (Гб)	4	4	4
Максимальный объем дисков (Тб)	8.3	8.3	20
SPECint95	11.8 (1ЦП)	4.6 (1ЦП)	5.34 (1ЦП)
SPECfp95	20.2 (1ЦП)	8.2 (1ЦП)	4.35 (1ЦП)
SPECrate_int95	415 (4ЦП)	163 (4ЦП)	531 (12ЦП)
SPECrate_fp95	400 (4ЦП)	248 (4ЦП)	263 (12ЦП)

8.4. **12.4. Серверы компании IBM**

Несмотря на пережитые в последние годы финансовые трудности, корпорация IBM остается самой большой и мощной компьютерной компанией в мире. Ее присутствие ощущается буквально на всех направлениях развития компьютерного рынка: от чисто научно-исследовательских работ до разработки и производства кристаллов, от производства персональных компьютеров до создания самых мощных систем класса "мейнфрейм". В последние годы для компании характерно резкое расширение распространения программных продуктов собственного производства и сервисных услуг по созданию и поддержке крупных информационных систем.

Семейство RS/6000

Хотя архитектура RISC была разработана научным сотрудником компании IBM Джоном Куком в 1974 году, первая коммерческая RISC разработка компании (RT/PC) не была принята рынком. Однако появившиеся вслед за этим компьютеры с архитектурой RISC, представленные компаниями Sun Microsystems, Hewlett-Packard и Digital Equipment были намного более успешными и действительно создали рынок RISC/UNIX рабочих станций и серверов. В феврале 1990 года, когда компания IBM поставила свою первую RISC-систему RS/6000, эти три поставщика прочно занимали лидирующие позиции.

В настоящее время семейство RS/6000 компании IBM включает в себя работающие под управлением операционной системы AIX рабочие станции и серверы, построенные на базе суперскалярной архитектуры POWER, расширенной архитектуры POWER2 и архитектуры PowerPC.

Процессоры POWER работают на частоте 33, 41.6, 45, 50 и 62.5 МГЦ. Архитектура POWER включает отдельные кэши команд и данных, 64- или 128-битовую шину памяти и 52-битовый виртуальный адрес. Она также имеет интегрированный процессор плавающей точки и таким образом хорошо подходит для приложений с интенсивными вычислениями, типичными для технической среды, хотя текущая стратегия RS/6000 нацелена как на коммерческие, так и на технические приложения. RS/6000 показывает хорошую производительность на плавающей точке: 134.6 SPECp92 для POWERstation/Powerserver 580. Первая реализация архитектуры POWER появилась на рынке в 1990 году. С тех пор компания IBM представила на рынок еще две версии процессоров POWER2 и POWER2+, обеспечивающих поддержку кэш-памяти второго уровня и имеющих расширенный набор команд. Эти процессоры работают с тактовой частотой 66.7 и 71.5 МГц и применяются в наиболее мощных рабочих станциях и серверах семейства RS/6000.

Для реализации быстрой обработки ввода/вывода в архитектуре POWER используется шина Micro Channel, имеющая пропускную способность 40 или 80 МГбайт/сек. Шина Micro Channel включает 64-битовую шину данных и обеспечивает поддержку работы нескольких главных адаптеров шины. Такая поддержка позволяет сетевым контроллерам, видеоадаптерам и другим интеллектуальным устройствам передавать информацию по шине независимо от основного процессора, что снижает нагрузку на процессор и соответственно увеличивает системную производительность.

Компания IBM распространяет влияние архитектуры POWER в направлении UNIX-систем с помощью платформы PowerPC. Архитектура POWER в этой форме обеспечивает уровень производительности и масштабируемость, превышающие возможности современных систем на платформе Intel. В архитектурном плане основные отличия этих двух разработок заключаются лишь в том, что системы PowerPC используют однокристальную реализацию архитектуры POWER, изготавливаемую компанией Motorola, в то время как большинство выпускавшихся до последнего времени систем RS/6000 использовали многокристальную реализацию. Сегодня уже выпускаются несколько вариаций процессора PowerPC, которые обеспечивают как потребности портативных изделий и настольных рабочих станций, так и достаточно мощных серверов. Первым на рынке появился процессор 601, первоначально предназначавшийся для использования в настольных рабочих станциях компаний IBM и Apple. На базе этого процессора уже построены многие модели рабочих станций и серверов RS/6000 (в том числе и реализующих симметричную мультипроцессорную обработку). Совсем недавно компания IBM представила рабочие станции моделей 42T и 42P, построенные на базе нового мощного процессора PowerPC 604, работающего с тактовой частотой 100, 120 и 133 МГц. В скором времени на рынке должен появиться микропроцессор PowerPC 620, который разработан специально для серверных конфигураций и ожидается, что со своей полностью 64-битовой архитектурой он обеспечит исключительно высокий уровень производительности.

В таблице 12.10 приведены основные характеристики серверов, а в таблице 12.11 многопроцессорных PowerPC SMP-серверов семейства RS/6000.

Таблица 12.11. Основные характеристики SMP-серверов RS/6000 на базе PowerPC

МОДЕЛЬ	
ЦП	
Тип процессора	
Тактовая частота (Мгц)	1
Число процессоров	
Системная шина (бит)	6
Размер кэша: –– L1 (команды/данные) (Кб) –– L2 (Мб)	16 0
ПАМЯТЬ	
Минимальный объем (Мб)	6
Максимальный объем (Мб)	10
ВВОД/ВЫВОД	
Количество слотов	5M
Периферийные интерфейсы	
Максимальная емкость внутренних дисков (Гб)	1
Максимальная емкость дисковой памяти (Гб)	3
ПРОИЗВОДИТЕЛЬНОСТЬ	
SPECint_rate95	3
SPECfp_rate95	2
SPECint_base_rate95	3
SPECfp_Base_rate95	2
tpmC	
$/tpmC	

8.4.1. 12.4.1. Модели C10 и C20 RISC System/6000

Эти системы представляют собой компактные серверы в конструктиве напольной тумбы. Они базируются на технологиях PowerPC и MicroChannel. В модели C10 применяется микропроцессор 80 МГц PowerPC 601, а в модели C20 - более мощный микропроцессор 120 МГц PowerPC 604. В обеих системах для увеличения производительности оперативной обработки транзакций предусмотрена возможность установки кэш-памяти второго уровня емкостью 1 Мбайт. Оперативная память, защищенная кодами ECC, может расширяться от 16 до 256 Мбайт, емкость внутренних дисков может достигать 6.6 Гбайт, а при использовании внешних накопителей - 654 Гбайт.

4 гнезда расширения MicroChannel (3 - в модели C10) обеспечивают подключение широкой номенклатуры адаптеров и периферийных устройств. Серверы работают под управлением ОС AIX версий 3.2.5 и 4, которая обеспечивает возможность эксплуатации большинства приложений, разработанных для других систем RS/6000 (построенных на базе процессоров POWER, POWER2 и PowerPC). Компания IBM рекомендует использовать эти системы в небольших компаниях в качестве сетевых серверов, а также в больших корпорациях в качестве серверов рабочих групп и отделов для реализации серверов приложений.

8.4.2. **12.4.2. Серверы серии 500 RISC System/6000**

Серверы серии 500 базируются на комбинации многокристальной технологии POWER2 и шинной архитектуре MicroChannel. Эти однопроцессорные системы обеспечивают высокий уровень производительности при решении вычислительных задач и задач оперативной обработки транзакций.

Оперативная память всех моделей серии 500 может расширяться от 64 Мбайт до 2 Гбайт. Объем внутренней дисковой памяти серверов может достигать 27 Гбайт, а с учетом возможности подключения внешних RAID-систем - до 2 Тбайт. В каждой системе предусмотрены 7 высокоскоростных гнезд расширения MicroChannel, обеспечивающих подключение широкой номенклатуры адаптеров и периферийных устройств.

Использование специального пакета HACMP/6000 for AIX дает возможность создания кластера, объединяющего до 8 систем серии 500.

В состав модели 591 входит микропроцессор 77 МГц POWER2, интегрированный сдвоенный процессор плавающей точки и четырехсловная система памяти с многоразрядной шиной, обеспечивающая быструю пересылку больших блоков данных. В эту систему включены также 16-битовый контроллер Fast/Wide SCSI-2, дисковый накопитель объемом 2 Гбайт и 4-скоростной считыватель компакт-дисков.

Модель 590 отличается от модели 591 лишь использованием менее скоростного процессора (66 МГц POWER2), что обеспечивает более низкую стоимость системы.

В стандартную поставку модели 59H входит кэш-память второго уровня емкостью 1 Мбайт. Эта система обеспечивает наивысшую производительность в серии 500 при выполнении приложений оперативной обработки транзакций.

8.4.3. **12.4.3. Модели G40 Server RS/6000**

Модель G40 считается системой начального уровня среди симметричных мультипроцессорных систем, предлагаемых компанией IBM. Она может включать от 1 до 4 процессоров PowerPC 604, оснащаться оперативной памятью объемом 64 Мбайт - 1 Гбайт, кэш-памятью второго уровня емкостью 512 Кбайт на каждый процессор (микропроцессор PPC 604 имеет встроенные раздельные кэши первого уровня для команд и данных объемом по 16 Кбайт). Для снижения задержек и увеличения пропускной способности подсистемы памяти IBM применяет архитектуру неблокируемого коммутатора данных. Максимальная полоса пропускания подсистемы памяти составляет 1.8 Гбайт/с.

В состав системы входит специальный сервисный процессор IBM SystemGuard, который постоянно наблюдает за состоянием системы, обеспечивает выполнение локальной и удаленной диагностики системы и поддерживает процессы реконфигурации системы в случае проявления неисправностей. Сервисный процессор выполняет рутинные операции по обслуживанию системы: включение и выключение питания, выполнение диагностических процедур и поддержку системной консоли. При обнаружении каких-либо проблем сервисный процессор выполняет необходимые действия по автоматическому восстановлению системы. Если по каким-либо причинам систему не удается перезапустить, SystemGuard может автоматически связаться с сервисными службами IBM.

Система поддерживает два высокоскоростных (160 Мбайт/с) канала ввода/вывода и 5 гнезд расширения MicroChannel, обеспечивающих подключение множества периферийных адаптеров и устройств.

В состав стандартной конфигурации системы входят: дисковый накопитель емкостью 2.2 Гбайт, 4-скоростной считыватель компакт-дисков, флоппи-дисковод, 5 гнезд расширения MicroChannel и 3 отсека для установки внутренних накопителей. В дополнительную стойку расширения могут устанавливаться до 6 накопителей. Внешняя память может быть организована как на базе технологии SCSI-2, так и на основе SSA (Serial Storage Architecture). Объем дисковой памяти при использовании четырех стоек расширения может достигать 121.5 Гбайт, а при использовании пяти дисковых подсистем IBM 7134 Model 010 - 350 Гбайт.

Серверы G40 работают под управлением операционной системы AIX версий 4.1 и 4.2. С целью создания систем высокой готовности до 8 серверов G40 могут объединяться в кластер. Для этого необходимо приобретение специального слоя программных средств HACMP - High Availability Cluster Multi-Processing for AIX.

8.4.4. 12.4.4. Модели J40 Server RS/6000

Среди симметричных мультипроцессорных систем, предлагаемых компанией IBM, J40 считается системой среднего уровня. Она может включать от 2 до 8 процессоров PowerPC 604, оснащаться оперативной памятью объемом 128 Мбайт - 2 Мбайт, кэш-памятью второго уровня емкостью 1 Мбайт на каждый процессор (микропрцессор PPC 604 имеет встроенные раздельные кэши первого уровня для команд и данных объемом по 16 Кбайт). Для снижения задержек и увеличения пропускной способности подсистемы памяти как и в системах G40 применяется архитектура неблокируемого коммутатора данных. Максимальная полоса пропускания подсистемы памяти составляет 1.8 Гбайт/с.

Как и в серверах G40 в состав системы входит специальный сервисный процессор IBM SystemGuard, обеспечивающий выполнение диагностических процедур, наблюдение за состоянием и реконфигурацию системы.

Система поддерживает два высокоскоростных (160 Мбайт/с) канала ввода/вывода и 14 гнезд расширения MicroChannel, обеспечивающих подключение множества периферийных адаптеров и устройств.

В состав стандартной конфигурации системы входят: дисковый накопитель емкостью 4.5 Гбайт, 4-скоростной считыватель компакт-дисков, 7 отсеков для установки дисковых и 3 для установки ленточных накопителей. В дополнительную стойку расширения J01, имеющую 8 гнезд расширения MicroChannel, могут устанавливаться 12 дисковых и 2 ленточных накопителя. Таким образом, объем внутренней дисковой памяти может достигать 36 Гбайт, плюс 99 Гбайт может быть размещено в стойке расширения J01. Следует отметить, что конструкция системы допускает "горячую" замену всех дисковых и ленточных накопителей без выключения питания системы. Внешняя память может быть организована как на базе технологии SCSI-2, так и на основе SSA (Serial Storage Architecture).

Серверы J40, также как и серверы G40, работают под управлением операционной системы AIX версий 4.1 и 4.2. С целью создания систем высокой готовности до 8 серверов J40 могут объединяться в кластер. Для этого необходимо приобретение специального слоя программных средств HACMP - High Availability Cluster Multi-Processing for AIX.

8.4.5. 12.4.5. Системы SP1 и SP2

Для технических приложений, предъявляющих очень большие требования к производительности, IBM также разрабатывает ряд высокопроизводительных систем с параллельной архитектурой. Первая из них, SP1, разработана прежде всего для задач с большим объемом вычислений таких как сейсмические исследования, гидродинамика и вычислительная химия. Она может иметь от 8 до 64 процессорных узлов POWER, обеспечивая производительность от 1 до 8 Гфлопс. Система поддерживает новейшую многоступенчатую архитектуру с пакетным переключением и может выполнять задачи как на отдельных процессорах, так и распараллеливать выполнение одной задачи на нескольких процессорах. Одним из узких мест SP1 является ввод/вывод, осуществляющийся через сеть Ethernet, и относительно небольшой объем дисковой памяти. Именно это и ограничивает ее применение для решения задач коммерческого назначения. С целью устранения этого узкого места IBM разработала следующий вариант этой системы, получивший название SP2. Она включает уже 128 узлов с процессорами POWER2, соединенных между собой высокоскоростным коммутатором с пропускной способностью 2.56 Гбайт/с. Система может поддерживать в каждый момент времени до 64 соединений, так что скорость обмена данными между ее двумя узлами достигает 40 Мбайт/с. Кроме того, к ней может теперь подключаться большое число внешних дисковых накопителей. На весенней выставке CeBIT в Ганновере, Германия, работа SP2 демонстрировалась не только при решении научно-инженерных задач типа прогноза погоды и расчетов в области географии, но и при работе таких коммерческих приложений как АСУ R3 компании SAP, CICS/6000 и DB2/6000 компании IBM. С теоретической точки зрения все, что разрабатывается для кластерных систем, сможет работать на системе SP2.

8.5. 12.5. Серверы компании Silicon Graphics

Компания Silicon Graphics (SGI) была создана в 1981 году. Основным направлением работы компании в течение многих лет было создание высокопроизводительных графических рабочих станций. В настоящее время ее интересы распространяются на рынок высокопроизводительных вычислений, как для технических, так и для коммерческих приложений. В частности она концентрирует свои усилия на разработке и внедрении современных технологий визуализации вычислений, трехмерной графики, обработки звука и мультимедиа.

Отчет независимой аналитической организации IDC 1995 года показал, что в период 1994 календарного года SGI оказалась самой быстро растущей компанией-поставщиком UNIX-систем. Правда она еще далеко отстает от лидеров (HP и Sun) по общему количеству поставок систем и доходу. (Например, в течение этого же периода SGI осуществила поставку 65000 систем, в то время как Sun увеличила на это число количество своих поставок). Silicon Graphics сохраняет лидирующие позиции на рынке суперкомпьютеров отделов, занимая 36% этого рынка и более чем вдвое опережая своего ближайшего конкурента. Недавно она приобрела известную компанию Cray Research и теперь расширяет свое присутствие на рынке суперкомпьютеров.

Разработка процессора R10000 позволила компании перейти к объединению своих серверов Challenge (на базе процессора R4000) и PowerChallenge (на базе процессора R8000) в единую линию изделий. Благодаря повышенной производительности этого процессора на целочисленных операциях и плавающей точке обе линии продуктов могут быть слиты без потери производительности.

Все высокопроизводительные серверы SGI обладают рядом средств по поддержанию свойств RAS (надежности, готовности и удобства обслуживания), в частности, обеспечивают автоматическое восстановление системы после сбоя и подключение источников бесперебойного питания.

Серверы Silicon Graphics работают под управлением операционной системы IRIX - ОС UNIX реального времени, построенной в соответствии с требованиями стандартов SVID (System V Interface Definition) и XPG4. Она поддерживает возможность работы нескольких машин на одном шлейфе SCSI (multi-hosted SCSI), 4-кратное зеркалирование и 128-кратное расщепление дисковых накопителей. На платформе поддерживаются многие продукты компаний Oracle, Informix и Sybase.

Основные характеристики серверов компании Silicon Graphics приведены в таблице 12.12.

Таблица 12.12.

МОДЕЛЬ Challenge	**S**
ЦП	
Тип процессора	MIPS R440 MIPS R460 MIPS R500
Тактовая частота (МГц)	200
Число процессоров	1
Системная шина (Мбайт/с)	267
Разрядность шины (бит) данных/адреса	-
Размер кэша:	
–– L1 (команды/данные)(Кб)	16/16 (R4400,R 32/32 (R500
–– L2 (Мб)	1 (R4400)

МОДЕЛЬ Challenge	S
	0.5 (R5000
ПАМЯТЬ ECC	
Минимальный объем (Мб)	32
Максимальный объем (Гб)	0.256
Расслоение	1,2
ВВОД/ВЫВОД	
Тип фирменной шины	SGI GIO
Пропускная способность(Мб/с)	267
Количество шин	1
Количество слотов	2 GIO
Тип стандартной шины	-
Пропускная способность стандартной шины (Мб/с)	-
Количество стандартных шин	-
Количество слотов стандартной шины	-
Периферийные интерфейсы	Fast/Wide SCSI-2
Максимальная емкость дисковой памяти	277 Гб

8.5.1. 12.5.1. Challenge S

В семействе серверов Challenge системы Challenge S представляют собой однопроцессорные компьютеры с достаточно высокой производительностью и хорошими возможностями для расширения. Архитектура этих серверов рассчитана на использование различных типов процессоров MIPS (в настоящее время используются процессоры R4400, R4600 и R5000) и высокоскоростной системной шины с пропускной способностью 267 Мбайт/с.

В стандартной конфигурации серверы Challenge S оснащаются двумя портами Ethernet, одним интерфейсом 10 Мбайт/с Fast SCSI-2 и двумя каналами Fast/Wide Differential SCSI-2, работающими со скоростью 20 Мбайт/с. Дальнейшее расширение возможно с помощью двух гнезд GIO, которые поддерживают дополнительные адаптеры SCSI, Ethernet, FDDI, ATM и видеоподсистемы. Внешняя дисковая память наращивается с помощью специальных шасси CHALLENGE Vault L, обеспечивающих 72 Гбайт дискового пространства. Максимальный объем дисков может достигать 277 Гбайт с использованием нескольких таких устройств.

8.5.2. 12.5.2. Challenge DM

Сервер сетевых ресурсов Challenge DM представляет собой систему начального уровня в семействе симметричных высокопроизводительных мультипроцессорных систем компании Silicon Graphics. Эти системы, прежде всего, нацелены на рынок баз данных, файловых серверов, серверов WWW, цифровой обработки данных и систем реального времени.

В архитектурном плане Challenge DM базируется на высокоскоростной системной шине с пропускной способностью в установившемся режиме 1.2 Гбайт/с и микропроцессорах MIPS R4400. Система поддерживает от 1 до 4 R4400, от 1 до 4 Мбайт кэш-памяти второго уровня на каждый процессор, до 6 Гбайт оперативной памяти и до трех подсистем ввода/вывода POWER Channel-2 с пропускной способностью 320 Мбайт/с.

В стандартной конфигурации серверы оснащаются одной подсистемой в/в POWER Challenge-2. Каждая такая подсистема включает контроллер Ethernet, два контроллера Fast/Wide 16-бит SCSI-2 с максимальной скоростью 20 Мбайт/с, два последовательных порта 19.2 Кбит/с RS232, один последовательный порт 38.4 Кбит/с RS422, параллельный порт и поддерживает работу до двух

дополнительных дочерних плат (модулей) фирменного интерфейса в/в - HIO. Внешняя память систем может быть сформирована с помощью устройств CHALLENGE Vault L, CHALLENGE Vault XL и CHALLENGE RAID, которые обеспечивают наращивание емкости дисков до уровня 3.7 Тбайт без использования средств RAID и до 10 Тбайт RAID-памяти.
Для расширения возможностей серверов можно использовать адаптеры стандартной шины VME-64 и модули HIO. Дополнительно в серверы могут быть установлены адаптеры HiPPI, ATM, FDDI и 8-портовый адаптер Ethernet.
Эти серверы оснащаются инструментальными средствами системного администрирования, а также средствами надежного резервного копирования данных и управления внешней памятью. В качестве дополнительной возможности предлагаются средства обеспечения высокой готовности системы.

8.5.3. 12.5.3. Challenge L

Серверы сетевых ресурсов Challenge L являются системами среднего класса в семействе симметричных высокопроизводительных мультипроцессорных систем SGI. Они также ориентированы на рынок мощных баз данных, файловых серверов, серверов WWW, цифровой обработки данных и систем реального времени.
В архитектурном плане они практически не отличаются от серверов Challenge DM, т.к. базируются на высокоскоростной системной шине с пропускной способностью в установившемся режиме 1.2 Гбайт/с. Но в качестве процессоров в них используются более современные микропроцессоры MIPS R10000, количество которых в системе может достигать 12. Система поддерживает от 2 до 12 процессоров R10000, от 1 до 4 Мбайт кэш-памяти второго уровня на каждый процессор, до 6 Гбайт оперативной памяти и до шести подсистем ввода/вывода POWER Channel-2 с пропускной способностью 320 Мбайт/с.
В стандартной конфигурации эти серверы имеют сходные характеристики с серверами Challenge DM, но имеют существенно большие возможности для расширения. В частности, внешняя память систем может быть сформирована с помощью устройств CHALLENGE Vault L, которые обеспечивают наращивание емкости дисков до уровня 5.6 Тбайт без использования средств RAID и до 17.4 Тбайт RAID-памяти.
Эти серверы также оснащаются мощными инструментальными средствами системного администрирования, а также средствами надежного резервного копирования данных и управления внешней памятью. В качестве дополнительной возможности предлагаются средства обеспечения высокой готовности системы.

8.5.4. 12.5.4. Challenge XL

Серверы сетевых ресурсов Challenge XL представляют собой наиболее мощные симметричные высокопроизводительные мультипроцессорные системы компании SGI. По своим возможностям они сравнимы с компьютерами класса "мейнфрейм" и могут работать с базами данных огромных размеров.
Они также базируются на высокоскоростной системной шине с пропускной способностью в установившемся режиме 1.2 Гбайт/с и микропроцессорах MIPS R10000, но количество ЦП в этих системах может достигать 36. Система поддерживает от 2 до 36 процессоров R10000, от 1 до 4 Мбайт кэш-памяти второго уровня на каждый процессор, до 16 Гбайт оперативной памяти и до шести подсистем ввода/вывода POWER Channel-2 с пропускной способностью 320 Мбайт/с. Таким образом, основным свойством этих систем является масштабируемость за счет увеличения числа процессоров, объемов оперативной памяти и подсистем ввода/вывода.

8.5.5. 12.5.5. Challenge DataArray

Чтобы обеспечить еще большее масштабирование своих систем SGI сравнительно недавно выпустила на рынок продукт под названием Challenge DataArray. По существу он представляет собой слабо связанную систему, построенную на базе высокоскоростных 16-портовых коммутаторов, позволяющих объединить до восьми серверов Challenge DM, L или LX, и набор инструментальных средств по администрированию системы и управлению ее производительностью. Эти системы поддерживают популярные параллельные СУБД Informix Extended Parallel Server (XPS) и Oracle Parallel Server (OPS) и предназначены для построения крупномасштабных систем принятия решений.

Основу для построения системы дает стандартный Высокопроизводительный Параллельный Интерфейс (HiPPI - High Performance Parallel Interface), обеспечивающий передачу данных со скоростью до 100 Мбайт/с (для сравнения стандарт FDDI обеспечивает скорость 12,5 Мбайт/с). Это 32-битовая параллельная реализация, рассчитанная на соединения точка-точка. HiPPI обеспечивает высокую производительность сетевых приложений благодаря 8-кратному увеличению скорости передачи (по сравнению с FDDI) и возможности передавать пакеты большего размера, оптимальные для организации пересылок память-память. Система строится на 1-2 неблокируемых 16-портовых HIPPI-коммутаторах. В отличие от Ethernet, Token Ring или FDDI, HiPPI не использует общую среду передачи данных. Кабели, соединяющие два интерфейса HiPPI, содержат пакеты, передаваемые источником данных в симплексном режиме. Эти пакеты могут быть видны на промежуточных коммутаторах, но недоступны для других узлов системы. После пересылки пакета соединение может быть закрыто или остаться открытым для пересылки дополнительных пакетов. HiPPI имеет раздельное управление соединением, пакетами и передачей данных.

Системы Challenge поддерживают от 1 до 4 HiPPI-связей, каждая из которых имеет полосу пропускания в установившемся режиме 92 Мбайт/с.

В качестве системной консоли в Challenge DataArray используется рабочая станция Indy с 16-портовым мультиплексором последовательных портов. Программное обеспечение включает инструментальные средства администрирования и управления системой, высокопроизводительный драйвер, управляющий работой нескольких HiPPI-связей и интерфейс прикладного уровня для Oracle OPS и Informix XPS.

8.6. **12.6. Серверы компании Sun Microsystems**

Компания Sun Microsystems основана в 1982 году и к настоящему времени является одной из трех крупнейших компаний-поставщиков на рынке высокопроизводительных рабочих станций и серверов.

С самого начала своей деятельности Sun придерживалась концепции проектирования открытых систем. Ее первые изделия базировались на семействе микропроцессоров Motorola 680X0 и частично на микропроцессорах Intel и работали под управлением собственной версии операционной системы UNIX - SunOS. Компания разработала сетевой протокол NFS, который стал промышленным стандартом и используется практически всеми компаниями-производителями вычислительных систем. С 1987 года компания взяла на вооружение концепцию компьютеров с сокращенным набором команд (RISC), разработав собственную архитектуру SPARC - Scalable Processor Architecture.

В настоящее время семейство рабочих станций и серверов компании Sun Microsystems включает в себя однопроцессорные и многопроцессорные системы, построенные на базе собственной архитектуры SPARC RISC и работающие под управлением операционной системы Solaris (SunOS 5).

Используя только процессоры SPARC в своих новейших рабочих станциях и серверах, Sun сохраняет контроль за развитием своей процессорной технологии, обеспечивает двоичную совместимость (прикладные пакеты, разработанные для одной рабочей станции SPARCstation или сервера, могут работать без каких-либо изменений на других рабочих станциях и серверах), а также масштабируемость (системы SPARC компании Sun обеспечивают ясную, модульную линию перехода от малых настольных станций до больших серверов).

Концепция открытых систем компании включает такие компоненты как: открытую архитектуру (SPARC и высокоскоростную шину S-bus; в настоящее время имеется 64-битовая версия S-bus), открытое системное программное обеспечение (UNIX System V Release 4, стандартные инструментальные средства и языки программирования, а также DOS), открытые интерфейсы прикладных программ (Open Look, Motif, X.11/NeWS, OpenFonts) и открытую среду сетевых вычислений (ONC+, NFS, SunLink и SunNet).

Компания обеспечивает средства связи своих систем с компьютерами класса mainframe и миникомпьютерами других фирм и работу в больших корпоративных средах. Операционная среда Solaris позволяет прикладным пакетам, ориентированным на UNIX, работать на аппаратных платформах SPARC, PowerPC и Intel (на этих платформах используется версия Solaris 2.5). Текущая версия Solaris 2.5 поддерживает симметричную многопроцессорную обработку, многопотоковую обработку, связанные с ней прикладные системы DeskSet и ToolTalk (объектно-ориентированную систему разработки, которая будет объединена с COSE, недавно согласованную общую операционную среду), обеспечивает поддержку NetWare и всех основных стандартов SVR4 (POSIX, XPG/3 и SVID).

Компания Sun Microsystems поставляет широкий ряд серверов, способных удовлетворить потребностям самых взыскательных пользователей. Основными принципами компании при построении серверов являются:

- • Строгая приверженность промышленным стандартам и стандартам открытых систем, позволяющая легко осуществлять интеграцию и связь не только среди собственных изделий компании, но и с компьютерным оборудованием практически всех ведущих фирм.
- • Гибкость конструктивных решений, обеспечивающая простую и удобную линию наращивания производительности путем увеличения количества процессоров, объемов оперативной памяти, емкости дисков, количества и типов сетевых адаптеров.
- • Двоичная совместимость, начиная с простых настольных систем и кончая высокопроизводительными центрами обработки данных, дающая возможность легко разрабатывать и переносить прикладные программы с системы на систему без дополнительных затрат.
- • Гибкие средства управления системой, упрощающие управление сложными сетями, включающими сети Novell NetWare, Banyan VINES или AppleTalk.
- • Средства высокой готовности, предусматривающие автоматическую реконфигурацию системы в случае отказа компонентов, использование дисковых массивов RAID и возможность построения кластеров, обеспечивающих работу с параллельными базами данных, а также снижающих неплановое и плановое время простоя систем.

Серверы рабочих групп строятся на базе SPARCserver 4, SPARCserver 5 и SPARCserver 20, выпускающихся в настольных конструктивах, совпадающих с конструкциями соответствующих рабочих станций. Характеристики этих серверов, а также характеристики серверов масштаба предприятия представлены в табл. 12.13-12.15.

8.6.1. **12.6.1. SPARCserver 4**

SPARCserver 4 - наиболее доступный по стоимостным характеристикам сервер, обладает прекрасным соотношением производительность/стоимость и может служить в качестве файл-сервера рабочей группы, принт-сервера или сервера приложений, а также в качестве сетевого сервера. Максимальный объем дисковой памяти, поддерживаемый сервером, составляет 28 Гбайт.

8.6.2. **12.6.2. SPARCserver 5**

SPARCserver 5 - один из наиболее популярных настольных серверов рабочих групп компании Sun. Его конструкция сбалансирована по производительности, использует S-bus - высокоскоростную шину ввода/вывода, являющуюся промышленным стандартом. Максимальный объем внешней дисковой памяти равен 100 Гбайт. В конфигурациях с названием Netra поставляется с программным обеспечением для объединения с сетями персональных компьютеров и сетью Internet.

8.6.3. **12.6.3. SPARCserver 20**

SPARCserver 20 предоставляет возможность создания многопроцессорных конфигураций. Компактная конструкция, высокая производительность при работе с базами данных и хорошие возможности интеграции в существующие локальные вычислительные сети позволяют создавать на базе SPARCserver 20 высокопроизводительные серверы рабочих групп, заменяющие существующие миникомпьютеры. Емкость массовой памяти, поддерживаемой на системах SPARCserver 20, может достигать 338 Гб.

8.6.4. **12.6.4. SPARCserver 1000/1000E**

Высокопроизводительный сервер SPARCserver 1000/1000E, который пришел на смену серверам серии 600MP, обеспечивает производительность на уровне больших вычислительных систем (mainframe) и может использоваться в качестве многоцелевого сервера NFS или сервера небольшого предприятия, поддерживающего работу до 1500 пользователей.

SPARCserver 1000/1000E имеет модульную конструкцию, в которую могут устанавливаться до 8 процессоров 50, 60 или 75 МГц SuperSPARC с кэшем второго уровня емкостью 1 Мб на каждый процессор. Для обеспечения эффективной работы в многопроцессорном режиме в SPARCserver 1000/1000E используется внутренняя шина XDbus с пропускной способностью 250 Мб/сек. Шина XDbus, работающая в режиме пакетного переключения, обеспечивает модульное расширение количества процессоров, карт ввода/вывода, оперативной памяти и S-bus слотов. В системном блоке могут устанавливаться до четырех плат (системных или дисковых). На каждой системной плате могут размещаться один или два процессорных модуля, работающих на частоте 50, 60 или 85 МГц и до 512 Мб оперативной памяти. Каждая системная плата содержит 3 слота расширения S-bus, два последовательных порта, контроллер SCSI-2 и контроллер сети Ethernet. Таким образом, максимальный объем оперативной памяти системы может достигать 2 Гб, а количество S-bus слотов - 12. Свободные слоты S-bus могут использоваться для установки карт видеоадаптеров, сетевых контроллеров и дополнительных контроллеров SCSI-2. Максимальный объем дискового пространства может достигать 760 Гб.

Работая под управлением операционной системы Solaris 2.5 (обеспечивает симметричную многопроцессорную обработку), SPARCserver 1000 в восьмипроцессорной конфигурации имеет отличную производительность при работе с базами данных - 400 TPC-A (транзакций в минуту) и соотношение стоимость/производительность - $5987/TPC-A.

8.6.5. **12.6.5. SPARCcenter 2000/2000E**

Многопроцессорный сервер SPARCcenter 2000 на сегодняшний день является достаточно мощным сервером компании Sun Microsystems. Это изделие представляет собой дальнейшее развитие стратегической линии компании на замену старых больших ЭВМ (mainframe) в учреждениях и организациях. В SPARCcenter 2000 могут устанавливаться от двух до двадцати процессоров SuperSPARC, что обеспечивает пиковую производительность до 2.7 GIPS (миллиардов операций в секунду) и до 350 MFLOPS (миллионов команд с плавающей точкой в секунду). Каждый процессор в системе снабжается внешним кэшем емкостью 2 Мб, а объем оперативной памяти системы может достигать 5 Гб. Для обеспечения эффективной работы 20 процессоров используется внутренняя сдвоенная шина SDbus с пропускной способностью 500 Мб/сек. Симметричная многопроцессорная архитектура сервера обеспечивает поддержку до 40 S-bus слотов расширения, дисковое пространство объемом до 4.8 Тбайт и предоставляет широкие возможности расширения для реализации очень больших баз данных. Благодаря реализации модульного принципа на всех уровнях конструктивной иерархии наращивание объема оперативной памяти, числа процессоров и дисков происходит значительно дешевле по сравнению с наращиванием аналогичных частей систем на базе миникомпьютеров и больших ЭВМ.

8.6.6. **12.6.6. SPARCcluster PDB server**

SPARCcluster PDB server - это название продукта, обеспечивающего объединение SPARCserver 1000 и SPARCcenter 2000 в системы высокой готовности, обеспечивающие поддержку Oracle Parallel Server.

8.6.7. **12.6.7. Ultra Enterprise 1**

Ultra Enterprise 1 представляет собой однопроцессорный сервер, построенный на базе микропроцессора UltraSPARC 1 с тактовой частотой 143 или 167 МГц в настольной конструкции. Объем кэш-памяти второго уровня составляет 512 Кб, оперативная память может расширяться от 32 Мб до 1 Гб, а максимальная емкость дисков может достигать 324 Гб.

8.6.8. **12.6.8. Ultra Enterprise 2**

Ultra Enterprise 2 - симметричный двухпроцессорный сервер в настольной конструкции, построенный на базе процессоров UltraSPARC 1 с тактовой частотой 167 или 200 МГц и объемом кэш-памяти второго уровня 512 Кб или 1 Мб. Максимальный объем оперативной памяти может достигать 2 Гб, а дисков 1 Тб. Сервер имеет 4 слота расширения периферийной шины SBus, 1 слот UPA и оснащается интерфейсом Fast/Wide SCSI-2 и контроллером 10/100 Мбит/с FastEthernet.

8.6.9. 12.6.9. Ultra Enterprise 3000 и Ultra Enterprise 4000

Семейство серверов Ultra Enterprise 3000, Ultra Enterprise 4000 представляют собой мощные, многоцелевые симметричные мультипроцессорные системы с повышенным уровнем готовности в компактной конструкции. Пропускная способность системной шины 2.5 Гбайт/с позволяет легко объединить до 6 или 12 процессоров (в системах 3000 и 4000 соответственно). Модульная конструкция позволяет легко варьировать количество процессорных плат и плат ввода вывода, а, следовательно, приспосабливать конфигурацию системы к конкретным нуждам заказчика. В системах 3000 объем оперативной памяти может достигать 6 Гбайт, максимальная емкость внутренних дисков - 42 Гбайт, а полная емкость поддерживаемой дисковой памяти превышать 2 Тбайта. В системах 4000 эти показатели равны соответственно 12 Гбайт, 16.8 Гбайт и более 4 Тбайт. Повышенный уровень готовности серверов обеспечивается возможностью замены большинства компонентов системы без выключения питания, применением методов избыточности в системах питания и охлаждения, а также специальным системным монитором SyMON, выполняющим постоянное диагностирование компонентов с целью предупреждения отказов системы.

8.6.10. 12.6.10. Ultra Enterprise 5000 и Ultra Enterprise 6000

Семейства серверов 5000 и 6000 являются очень мощными системами, обеспечивающими работу с огромными базами данных. Эти серверы оформлены в стоечной конструкции, но используют унифицированные с серверами 3000 и 4000 компоненты. Запас пропускной способности системной шины (2.5 Гбайт/с) позволяет строить системы с 14 (модели 5000) и 30 (модели 6000) процессорами. Максимальный объем оперативной памяти в системах 5000 составляет 14 Гбайт, максимальная емкость внутренних дисков 216 Гбайт, а общая дисковая память может достигать более 6 Тбайт. Для систем 6000 эти показатели составляют 30 Гбайт, 162 Гбайт и более 10 Тбайт соответственно. Пропускная способность платы ввода/вывода, которая включает две шины SBus, 3 слота SBus, контроллер FastEthernet, контроллер Fast/Wide SCSI и два гнезда волоконно-оптических каналов, составляет 200 Мбайт/с. Подбирая необходимое количество процессорных плат и плат ввода/вывода можно практически создать любую необходимую пользователю конфигурацию системы.

Таблица 12.13. Основные характеристики корпоративных серверов компании SUN Microsystems

МОДЕЛЬ	
ЦП	
Тип процессора	
Тактовая частота (МГц)	
Число процессоров	
Размер кэша (Кб) (в процессоре/на плате)	
Пропускная способность системной шины (Гб/сек)	
ПАМЯТЬ	
Минимальный объем (Мб)	
Максимальный объем (Гб)	
ВВОД/ВЫВОД	
Количество слотов	
Максимальная пропускная способность платы в/в (Мб/сек)	
Периферийные интерфейсы	
Максимальная емкость внутренних дисков (Гб)	
Максимальная емкость	

МОДЕЛЬ	
дисковой памяти (Тб)	
Сетевые интерфейсы основной/дополнительные	
ПРОИЗВОДИТЕЛЬНОСТЬ	
TPC-C	
$/tpmC	
NFS op/sec	
SPECrate_int92	
SPECrate_fp92	
AIM III (job/minute/ users)	

Таблица 12.14. Основные характеристики серверов рабочих групп компании SUN Microsystems

	SPARCserver 5		
	Model 110	**Model 71**	**Model712M**
	microSPARC II	SuperSPARC-II	
	110	75	75
	1	1	2
	64	64	64
	24	36/1024	36/1024 per CPU
Мб/сек)	105	105	105
	32	32	64
	256	512	512
	Sbus	Sbus	Sbus
	3	4	4
истемы в/в (Мб/сек)	52	52	52
	SCSI-2	SCSI-2	SCSI-2
	4.2	4.2	4.2
	118	339	339
	2	2	2
	1	1	1
	Ethernet/FDDI, ATM, Token Ring, FastEthernet	Ethernet/ FDDI, ATM, Token Ring, FastEthernet	Ethernet/ FDDI, ATM, FastEthern

	SPARCserver 5		
	Model 110	Model 71	Model712
	145	200	305
	1864	2984	5726
	1549	2875	5439
	1630	2761	5332
	1494	2595	4923

Таблица 12.15. Основные характеристики серверов отделов компании Sun Microsystems

	SPARCserver 1000E		
	SuperSPARC		
	85	85	8
	2	4	8
	64	64	6
	36/1024 per CPU	36/1024 per CPU	36/1 per C
	250	250	25
	32	64	6
	2048	2048	20
	Sbus	Sbus	Sb
	3-12	3-12	3-1
сек)	90	90	9
	SCSI-2	SCSI-2	SCS
	1050, 2100	1050, 2100	105 21
	764	764	76
	2-8	2-8	2-
	Ethernet/ FDDI,ATM, TokenRing, FastEthernet		
	-	-	104
	-	-	39
	5988	11508	217
	5805	11322	208
	5480	10557	202
	5232	9943	187
	2037/ 1849	3654/ 3327	606 53

Таблица 12.15. (Продолжение). Основные характеристики серверов отделов компании Sun Microsystems

МОДЕЛЬ	
ЦП	
Тип процессора	
Тактовая частота (МГц)	
Число процессоров	
Размер кэша (Кб) (в процессоре/на плате)	
Пропускная способность системной шины (Гб/сек)	
ПАМЯТЬ	
Минимальный объем (Мб)	
Максимальный объем (Гб)	
ВВОД/ВЫВОД	
Количество слотов	
Максимальная пропускная способность платы в/в (Мб/сек)	
Периферийные интерфейсы	
Максимальная емкость внутренних дисков (Гб)	
Максимальная емкость дисковой памяти (Тб)	
Сетевые интерфейсы основной/дополнительные	
ПРОИЗВОДИТЕЛЬНОСТЬ	
TPC-C	
$/tpmC	
NFS op/sec	
SPECrate_int92	
SPECrate_fp92	
AIM III (job/minute/ users)	

8.7. 12.7. Отказоустойчивые серверы компании Tandem Computer Inc.

8.7.1. 12.7.1. Введение

Надежные вычислительные машины являются ключевыми элементами для построения наиболее ответственных прикладных систем в сфере розничной торговли, финансов и телефонной коммутации. На современном этапе развития информационных технологий подобные приложения предъявляют широкий диапазон требований к масштабируемости, поддержке открытых стандартов и обеспечению отказоустойчивости систем. Одной из наиболее известных в мире фирм, работающих в данной области, является компания Tandem. В настоящее время для удовлетворения различных требований рынка надежных вычислений она поставляет две различные линии своих изделий: системы Tandem NonStop и системы Tandem Integrity.

Системы Tandem NonStop, первые модели которых появились еще в 1976 году, базируются на реализации многопроцессорной обработки и модели распределенной памяти. Для обеспечения восстановления после сбоев аппаратуры и ошибок программного обеспечения эти системы используют механизмы передачи сообщений между процессными парами. База данных NonStop SQL, в основе архитектуры которой лежит модель системы без разделения ресурсов (shared-nothing), показала линейную масштабируемость в приложениях обработки транзакций на конфигурациях, содержащих более 100 процессоров. Первоначально системы NonStop были нацелены на создание приложений оперативной обработки транзакций (OLTP), но в настоящее время интенсивно используются и в других ответственных приложениях (системах передачи сообщений и системах поддержки принятия решений).

Для удовлетворения потребностей рынка в отказоустойчивых системах, позволяющих выполнять без переделок существующие UNIX-приложения, в 1990 году компания Tandem объявила о начале выпуска систем Integrity. Для маскирования ошибок в работе систем Integrity используются методы аппаратной избыточности (трехкратное резервирование), обеспечивающие продолжение непрерывной работы в условиях сбоев без воздействия на приложения. Системы Integrity часто используются в телефонных и сотовых сетях связи, а также в других коммерческих приложениях, требующих реализации надежных систем, удовлетворяющих открытым стандартам.

Хотя указанные две линии изделий компании имеют отличия, они удовлетворяют целому ряду общих требований и используют многие общие технологии и компоненты. Все системы Tandem гарантируют целостность данных и устойчивость к сбоям, и кроме того, обеспечивают масштабируемость и возможность производить модернизацию системы в режиме online. Обе линии изделий NonStop и Integrity позволяют выполнять техническое обслуживание систем в режиме online (установку и замену плат, источников питания и вентиляторов без остановки системы и выключения питания). Применяемые конструкции допускают установку обеих систем в офисных помещениях, стандартных машинных залах вычислительных центров или на телефонных станциях. В системах используются много общих компонентов таких, как накопители на дисках, элементы памяти и микропроцессоры.

В 1991 году компания Tandem начала программу объединения лучших свойств обеих систем в единой линии изделий. Эта программа дает возможность гибкой реализации целого ряда важнейших свойств: устойчивости к сбоям (восстановление после проявления неисправности может выполняться как программными, так и аппаратными средствами), масштабируемости (построение кластеров на базе модели распределенной памяти и реализация мультипроцессорной обработки в разделяемой общей памяти) и использовании нескольких операционных систем (NonStop Kernel, Unix и Microsoft Windows NT).

Основой для объединения архитектур послужила разработка главного транспортного средства - системной сети ServerNet. ServerNet представляет собой многоступенчатую пакетную сеть, используемую как для организации межпроцессорных связей, так и для реализации связей с устройствами ввода/вывода. ServerNet обеспечивает эффективные средства для обнаружения и изоляции неисправностей, а также реализует прямую поддержку альтернативных каналов передачи данных для обеспечения непрерывной работы системы при наличии отказов сети. Разработка этой сети предоставляет новые возможности развития обеих линий изделий, включая большую масштабируемость, интерфейсы с открытыми стандартами шин и улучшенную поддержку мультимедийных приложений.

8.7.2. **12.7.2. Архитектура систем NonStop**

На рис. 12.6 показана базовая архитектура систем NonStop. Эта архитектура предполагает объединение двух или более ЦП при помощи дублированной высокоскоростной межпроцессорной шины. Каждый процессор имеет один или несколько каналов в/в, соединяющих его с двухпортовыми дисковыми контроллерами и коммуникационными адаптерами. В действительности в первых пяти поколениях систем NonStop (NonStop I, II, TXP, CLX и VLX) было реализовано только по одному каналу в/в на процессор, а пара разделяемых шин обеспечивала объединение до 16 процессоров. В более поздних системах NonStop Cyclone и Himalaya K10000/20000 для увеличения пропускной способности системы межсоединений была применена сегментация межпроцессорной шины на базе четырехпроцессорных секций. Секции могут объединяться с помощью оптоволоконных линий связи в узлы (до четырех секций в узле). Системы NonStop II, TXP, VLX и Cyclone поддерживают также возможность построения оптоволоконного кольца, которое позволяет объединить между собой до 14 узлов, и обеспечивает быстрый обмен данными внутри домена, состоящего из 224 процессоров. В системе Cyclone к каждому процессору могут подсоединяться несколько каналов в/в, причем каждые четыре канала управляются своей парой контроллеров прямого доступа к памяти.

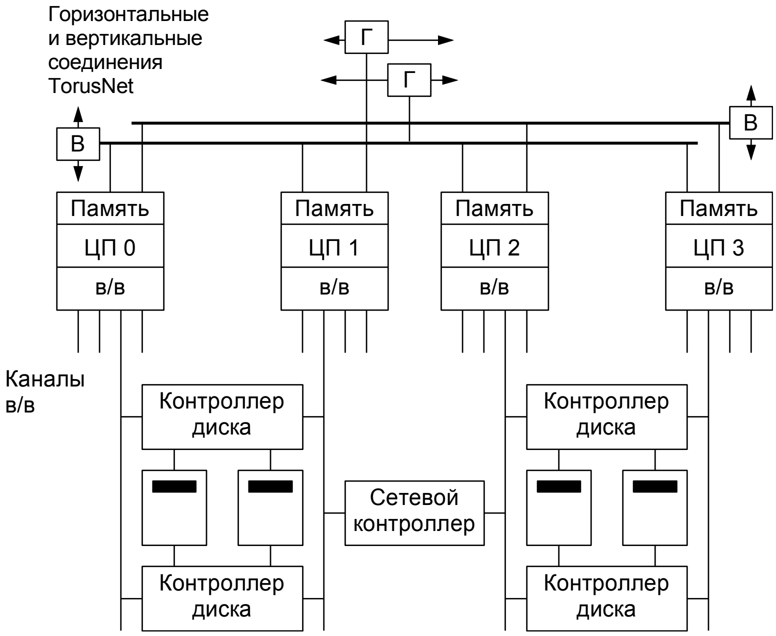


Рис. 12.6. Архитектура NonStop

После разработки и успешных испытаний системы Cyclone компания Tandem перешла на применение в своих изделиях RISC процессоров компании MIPS (вместо использовавшихся ранее заказных CISC процессоров). В системах CLX/R и K200 используются процессоры R3000, а в системах Himalaya K10000, K20000 и K2000 - процессоры R4400. Одновременно с объявлением в 1993 году о начале поставок нового семейства систем Himalaya компания анонсировала также оптоволоконную сеть межпроцессорного обмена TorusNet, предназначенную для построения крупномасштабных кластеров. TorusNet обеспечивает соединение четырехпроцессорных секций с помощью избыточной сети с топологией двухмерного тора.

Все аппаратные компоненты систем NonStop построены на основе принципа "быстрого проявления неисправности" (fail fast disign), в соответствии с которым каждый компонент должен либо функционировать правильно, либо немедленно останавливаться. В более ранних системах Tandem реализация этого принципа широко опиралась на использование методов проверки четности, избыточного кодирования или проверки допустимости состояния при выполнении каждой логической функции. Современные конструкции для обнаружения ошибок в сложной логике полагаются главным образом на методы дублирования и сравнения. Все системы, имеющие ЦП на базе микропроцессоров, для гарантии целостности данных и быстрого обнаружения неисправностей выполняют сравнение выходов дублированных и взаимно синхронизированных микропроцессоров. В системах NonStop ответственность за восстановление после обнаружения неисправности в аппаратуре возлагается на программное обеспечение.

Операционная система NonStop Kernel систем NonStop непрерывно развивалась и к настоящему времени превратилась из патентованной фирменной операционной системы в систему, которая обеспечивает полностью открытые интерфейсы, построенные на основе промышленных стандартов. Для поддержки устойчивости критически важных процессов в NonStop Kernel реализованы низкоуровневые механизмы контрольных точек, а также специальный слой программных средств, на котором строится как патентованная среда Guardian, так и открытая среда Posix-XPG/4. NonStop Kernel базируется на механизмах передачи сообщений и обеспечивает средства прозрачного масштабирования системы в пределах 16-процессорного узла, 224-процессорного домена или 4080-процессорной (локальной или глобальной) сети TorusNet.

8.7.3. **12.7.3. Архитектура систем Integrity**

Основной задачей компании Tandem при разработке систем семейства Integrity было обеспечение устойчивости к одиночным отказам аппаратуры при соблюдении 100% переносимости стандартных UNIX-приложений. Для маскирования аппаратных неисправностей в системах Integrity используется тройное модульное резервирование (TMR - triple-modular redundancy) в процессоре, кэш-памяти и основной памяти (см. рис. 12.7).

Три процессора выполняют одинаковые потоки команд, но работают с независимой синхронизацией. Процессоры синхронизируются во время обработки обращений к глобальной памяти и при обслуживании внешних прерываний. Все обращения к глобальной памяти с выходов резервируемых процессоров поступают через схемы голосования в пару контроллеров TMR. Схемы голосования на основе сравнения обращений между собой обнаруживают возможные неисправности процессоров и посылают достоверные запросы в остальную часть системы. Для обнаружения неисправностей в конструкциях контроллера TMR и процессора в/в используются средства самоконтроля. Каждый периферийный контроллер содержит стандартную плату VME, которая через специальную плату адаптера подсоединяется к паре шин в/в, защищенных четностью. Плата адаптера позволяет осуществлять коммутацию контроллера с двумя процессорами в/в.

В системах Integrity реализация платы основного процессора не требует сложной логики самоконтроля. Однако это делает ее конструкцию отличной от конструкции процессорной платы систем NonStop, хотя в обеих используются одни и те же микропроцессоры. Архитектура новых систем объединяет требования базовой конструкции Integrity при сохранении совместимости с требованиями систем NonStop.

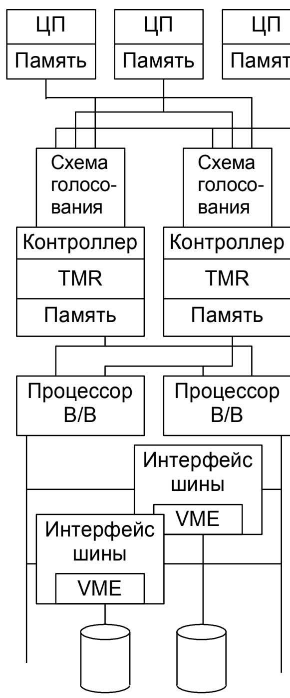


Рис. 12.7. Архитектура систем Integrity.

8.7.4. **12.7.4. Архитектура системы на базе ServerNet**

Новая системная архитектура, построенная на базе ServerNet, объединяет свойства систем NonStop и Integrity. Она решает общую задачу построения отказоустойчивых систем различного масштаба путем реализации гибких методов соединения стандартных функциональных блоков (модулей ЦП/памяти, подсистем внешней памяти и коммуникационных адаптеров).

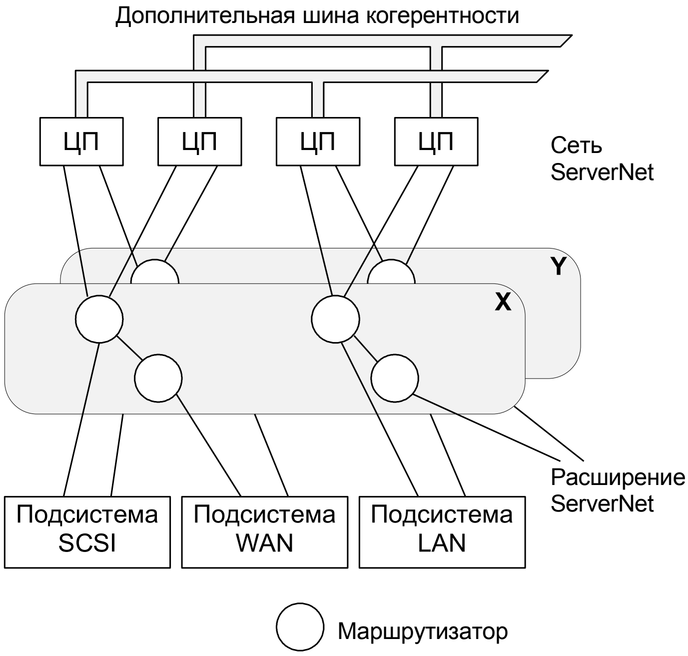


Рис. 12.8. Архитектура системы на базе ServerNet

На рис. 12.8 показана архитектура типичной системы, построенной на базе ServerNet. Эта система состоит из нескольких процессорных узлов и узлов ввода/вывода, объединенных друг с другом системной сетью ServerNet. Базовым элементом системной сети ServerNet является маршрутизатор, выполненный в виде отдельной заказной СБИС. Для обеспечения отказоустойчивости предполагается возможность построения двух независимых подсетей ServerNet: X и Y. В типовой конфигурации системы большинство ее узлов имеют двухпортовые интерфейсы, обеспечивающие подсоединение каждого узла к этим независимым подсетям. Одной из дополнительных возможностей новой архитектуры является наличие специальной шины когерентности, допускающей подключение до четырех ЦП. Эта шина обеспечивает согласованное состояние общей для нескольких процессорных узлов памяти и их кэшей при выполнении программ, разработанных в расчете на мультипроцессорную обработку в системе с разделяемой общей памятью.

При работе под управлением операционных систем, поддерживающих отказоустойчивость программными средствами (подобных NonStop Kernel), процессорные узлы выполняют независимые потоки команд. В отличие от более ранних систем, которые для передачи сообщений между процессорами и реализации операций ввода/вывода использовали разные интерфейсы, в новой архитектуре все пересылки данных осуществляются ЦП по сети ServerNet.

При использовании операционных систем, в которых отсутствуют специальные средства поддержки отказоустойчивости, последнее свойство может быть реализовано с помощью аппаратных средств путем создания конфигураций ЦП в виде дуплексных пар. В этом случае пары узлов ЦП выполняют идентичные потоки команд. Если один ЦП из пары отказывает, другой продолжает работать. Таким процессорам в сети ServerNet присваивается общий идентификатор узла, и все пакеты, адресуемые с помощью этого идентификатора, дублируются и доставляются одновременно двум ЦП. При отсутствии неисправностей оба ЦП в паре создают идентичные исходящие пакеты. Поэтому в случае нормальной работы логика маршрутизации ServerNet может выбрать для пересылки пакеты любого узла. При этом для обнаружения неисправностей используются возможности самой сети ServerNet.

Как уже отмечалось, для обеспечения отказоустойчивости в системе Integrity требуются три процессорных кристалла и три массива микросхем памяти. Новая архитектура требует четырех процессорных кристаллов (два на модуль ЦП) и двух массивов микросхем памяти. Стоимость реализации этих двух подходов существенно зависит от размера памяти. Для типовых систем оба метода имеют сравнимую стоимость.

8.7.4.1. ***12.7.4.1. ServerNet***

ServerNet представляет собой быструю, масштабируемую, надежную системную сеть, обеспечивающую гибкость соединения большого числа ЦП и периферийных устройств в/в между собой. Главными свойствами этой сети коммутации пакетов являются малая задержка и высокая надежность передачи данных. Для уменьшения задержки в сети применяется метод червячной маршрутизации, не требующий приема всего пакета до его отсылки к следующему приемнику. Физический уровень ServerNet образуют независимые каналы приема и передачи, каждый из которых имеет 9-битовое поле команд/данных и сигнал синхронизации. Поле команд/данных обеспечивает кодирование 256 символов данных и до 20 символов команд. Символы команд используются для управления уровнем звена, инициализации и сигнализации об ошибках. Кодирование в одних и тех же линиях команд и данных сокращает количество контактов и упрощает обнаружение ошибок в логике управления.

Система использует ServerNet для организации связей ЦП-ЦП, ЦП-В/В и В/В-В/В. Пересылки между микропроцессором и памятью для каждого узла ЦП остаются локальными.

Данные в сети ServerNet пересылаются со скоростью 50 Мбайт в секунду. Такая скорость передачи данных была выбрана исходя из того, чтобы превзойти потребности существующих периферийных устройств при соблюдении низких цен. В будущих поколениях ServerNet производительность линий связи будет увеличиваться по мере необходимости.

В настоящее время максимальная длина линии связи ServerNet ограничена 30 м. В будущих адаптерах предполагается увеличение расстояния между узлами ServerNet с помощью последовательных оптоволоконных линий связи. Предполагается, что этот переход будет относительно простым, поскольку все функции управления используют одни и те же линии команд/данных.

Пакеты ServerNet состоят из 8-байтного заголовка, дополнительного 4-байтного адреса, поля данных переменного размера и поля контрольной суммы. Заголовок определяет идентификаторы источника и приемника, объем передаваемых данных и необходимую для выполнения операцию. 4-байтное адресное поле обеспечивает 4-гигабайтное окно в адресном пространстве узла назначения. В устройствах в/в какие-либо механизмы преобразования адресов отсутствуют. ЦП преобразуют адрес из пакета в адрес физической памяти с помощью таблицы удостоверения и преобразования адреса (AVT - Address Validation and Translation Table). AVT проверяет допустимость удаленных обращений к локальной памяти узла. Каждый элемент таблицы обеспечивает доступ по чтению и/или записи одного узла ЦП или узла в/в к данным внутри 4-килобайтного диапазона адресов локальной памяти. С помощью нескольких элементов таблицы AVT узел ЦП может иметь доступ к большим сегментам своей памяти или обеспечивать несколько узлов доступом к одному и тому же сегменту памяти. Неисправный узел может испортить только те сегменты памяти, к которым ЦП предоставил ему доступ. AVT выполняет также преобразование смежных адресов памяти в несмежные адреса страниц физической памяти.

Все транзакции по сети ServerNet происходят в два этапа: выполнение запроса и ожидание соответствующего ответа, который должен вернуться до истечения заданного интервала времени (счетчика таймаута). Все узлы ServerNet поддерживают возможность выдачи несколько исходящих запросов в другие узлы. Помимо стандартных действий в соответствии с протоколом уровня передачи пакетов, получатель определенной транзакции может по своему усмотрению выполнить некоторые локальные действия. Например, существующие в настоящее время узлы ЦП или В/В преобразуют транзакции записи по определенным адресам в локальные прерывания.

В СБИС маршрутизатора ServerNet реализован матричный переключатель размерностью 6x6. Решение о направлении маршрутизации принимается на основе анализа идентификатора приемника из заголовка пакета. В состав маршрутизаторов входят входные буфера FIFO, логика арбитража и управления потоком данных, реализованная с помощью ЗУПВ таблица маршрутизации и матричный переключатель (см. рис. 12.9). Инициализация и реконфигурация сети выполняются программными средствами путем загрузки соответствующих таблиц маршрутизации.

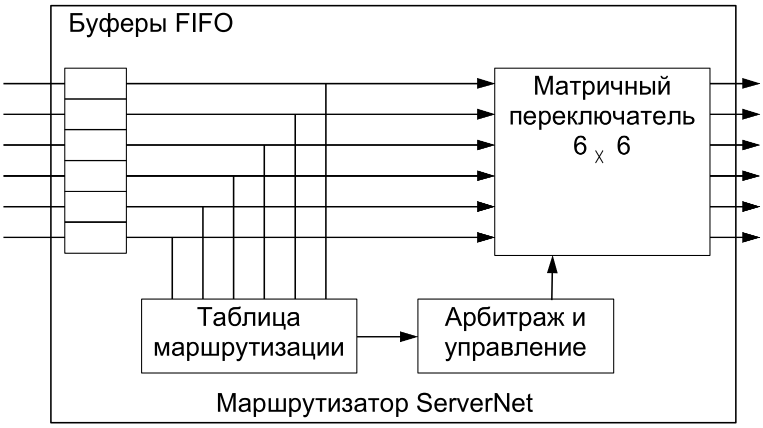


Рис. 12.9. СБИС 6-портового маршрутизатора ServerNet.

8.7.4.2. ***12.7.4.2. Процессорный модуль***

Одним из базовых элементов системы является процессорный модуль (ЦП), блок-схема которого показана на рис. 12.10. В ЦП, построенном на принципах быстрого проявления неисправностей, имеются два порта ServerNet, обеспечивающие его соединение через системную сеть с другими ЦП и устройствами в/в. Для реализации механизмов разделяемой общей памяти несколько ЦП могут объединяться друг с другом с помощью шины когерентности.

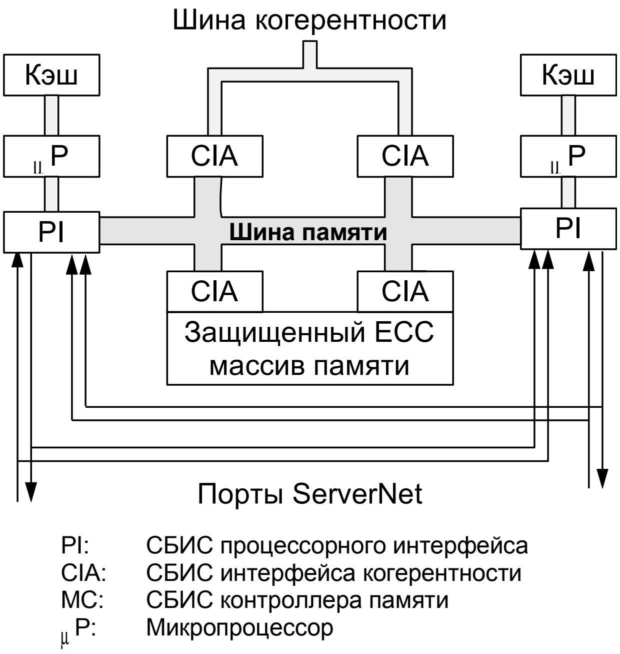


Рис. 12.10. Блок-схема ЦП.

В состав ЦП входят два микропроцессора, каждый из которых имеет независимый вторичный кэш. Каждый микропроцессор подсоединяется к шине памяти с помощью СБИС процессорного интерфейса. При выполнении обращений к памяти эти СБИС сравнивают выходы обоих микропроцессоров для обнаружения всех возможных ошибок микропроцессора и кэша. Память ЦП защищена кодом ЕСС, который обеспечивает коррекцию всех одиночных ошибок и обнаружение любых ошибок в отдельных микросхемах ДЗУПВ или адресных линиях. Массив микросхем памяти соединяется с шиной памяти ЦП через пару СБИС контроллеров памяти. Эти СБИС во время обращений к памяти взаимно проверяют выходы друг друга.

Как уже было отмечено, ЦП могут иметь прямой доступ к памяти других ЦП с помощью дополнительной шины когерентности. Эта шина обеспечивает аппаратную поддержку стандартных приложений UNIX или Windows NT, которые используют симметричную мультипроцессорную обработку (SMP). Каждый ЦП подсоединяется к шине с помощью пары самоконтролирующихся СБИС интерфейса когерентности. Эти СБИС обеспечивают кэш-когерентный доступ к общей памяти используя дублированную память тегов и стандартный протокол аннулирования блоков кэш-памяти. Они обеспечивают также когерентность кэш-памяти при выполнении обращений к памяти со стороны В/В. Все передачи данных по шине когерентности защищены кодом ECC. Проверка синдрома ECC для данных, пересылаемых по шине, и сравнение выходов СБИС позволяет обнаруживать сбои шины или СБИС интерфейса.

Операционная система, поддерживающая модель системы без разделения ресурсов (подобная NonStop Kernel), для увеличения степени изоляции ошибок может запретить работу с шиной когерентности. В этом режиме когерентность кэш-памяти для всех транзакций с памятью по сети ServerNet обеспечивается средствами системного программного обеспечения. Если же работа шины когерентности разрешена, то, вообще говоря, ошибка в одном ЦП может привести к отказу всех ЦП, подсоединенных к этой шине.

СБИС процессорного интерфейса ЦП реализуют два порта ServerNet. Линии приема данных обоих портов ServerNet подсоединяются к обеим СБИС процессорного интерфейса. Каждая СБИС формирует данные для передачи по обоим портам ServerNet, но реально данные передаются только из одного порта. Вторая СБИС принимает данные, передаваемые другой СБИС, сравнивает полученное значение со значением, которое она сформировала сама, и сигнализирует об ошибке при любом рассогласовании данных.

8.7.4.3. ***12.7.4.3. Организация ввода/вывода***

Новая система в/в обеспечивает практически неограниченные возможности масштабирования по числу объединяемых узлов и пропускной способности сети. Она эффективно поддерживает модель распределенных вычислений разрешая любому процессору обращаться к любому контроллеру в/в и допуская реализацию прямых связей контроллер-контроллер.

Большинство адаптеров в/в ServerNet также имеют по два порта. Эти порты обеспечивают две независимые магистрали доступа к адаптеру. Возникновение ошибки в одной из магистралей ServerNet обнаруживается средствами временного контроля. Системное программное обеспечение позволяет перемаршрутизировать и заново повторить запрос.

8.7.4.4. ***12.7.4.4. Дуплексная работа***

Аппаратная отказоустойчивая система реализуется с помощью дуплексной пары, которая создается путем соответствующего конфигурирования двух процессорных модулей. Идентичное состояние памяти и кэшей в этих двух модулях поддерживается благодаря выполнению на обоих ЦП одного и того же программного кода с теми же самыми данными, а также поступлению в память обоих ЦП всего потока ввода. Оба ЦП генерируют идентичные исходящие потоки вывода. Один из этих потоков выбирается маршрутизаторами для пересылки в контроллеры в/в или другие процессоры.

Дуплексная работа затрагивает три аспекта построения системы: межмодульную синхронизацию, синхронизацию уровня линий связи ServerNet и обработку ошибок. Для реализации дуплексного режима требуются два маршрутизаторов в различных подсетях ServerNet и два ЦП, подсоединенные к разным шинам когерентности (для работы с разделяемой памятью). Отказ одного из компонентов системы не может вывести из строя оба ЦП или оба маршрутизатора. Все ЦП, разделяющие шину когерентности, имеют общую синхронизацию. Специалисты Tandem называют эту комбинацию процессорных модулей (ЦП), шины когерентности и системы синхронизации слайсом. Система, сконфигурированная для дуплексной работы, имеет два слайса.

Работа дуплексной системы требует синхронной работы двух слайсов. ЦП в каждом слайсе в одном и том же такте выполняют идентичные команды. Входной поток ввода поступает в интерфейс памяти каждого ЦП в одном и том же такте. Поэтому в нормальных условиях при отсутствии неисправностей поведение слайсов будет полностью идентичным.

Модули ЦП имеют развитые средства обнаружения неисправностей. ЦП останавливается при обнаружении его схемами контроля любой ошибки. Остановка ЦП приводит к тому, что по обоим его портам ServerNet будет передана запрещенная кодовая комбинация. В результате маршрутизатор может определить неисправный ЦП (основополагающим правилом системы установлено, что все ошибки ЦП должны приводить к передачам по ServerNet запрещенных кодовых комбинаций).

Когда маршрутизатор, подсоединенный к дуплексному ЦП, обнаруживает ошибку, он начинает выполнение протокола восстановления. Этот протокол реализован полностью аппаратно без привлечения программных средств. При этом один из ЦП исключается из работы, а другой свою работу продолжит. Протокол гарантирует, что исправный ЦП останется работать. Однако существуют случаи, когда в исключенном ЦП неисправности отсутствуют. Например, к исключению ЦП из работы могут привести неисправности в одном из маршрутизаторов или в одной из линий связи ServerNet. В этих случаях система обслуживания может исключить из работы неисправный маршрутизатор, а исключенный ЦП перевести в состояние online.

Если при пересылке пакета из ЦП маршрутизатор обнаруживает неисправность линии связи ServerNet, он помечает пакет как недостоверный. Любой узел ServerNet, который получит этот пакет, будет его игнорировать. Это означает, что неисправность в ЦП, маршрутизаторе или линии связи может привести к потере одного или нескольких пакетов. При нормальной дуплексной работе только один из двух маршрутизаторов дуплексных процессоров пересылает пакеты, поступающие из каждого ЦП. Это ограничивает потерю пакетов пределами одной подсети ServerNet. Интерфейсные кристаллы обнаруживают потерю пакетов ServerNet с помощью средств временного контроля. Программное обеспечение ввода/вывода выполняет восстановление путем повторной передачи данных по альтернативному пути.

8.7.4.5. *12.7.4.5. Возможности масштабирования системы*

ServerNet обеспечивает широкие возможности для масштабирования системы. Обычно расширение выполняется с помощью встроенных кабельных соединений, а также установки в гнезда расширения ServerNet плат маршрутизаторов. Кроме того, добавление каждого ЦП обеспечивает увеличение числа линий связи ServerNet и эффективно расширяет общую пропускную способность в/в системы. В отличие от других массивно-параллельных архитектур сети ServerNet не ограничены только регулярными топологиями типа гиперкубов или торов. Сеть ServerNet позволяет увеличить число линий связи в любом месте, где требуется дополнительная пропускная способность. Приложения с умеренными требованиями к системе межсоединений могут довольствоваться минимальным количеством связей, а следовательно, использовать достаточно дешевую сеть, в то время как приложения с высокой интенсивностью обработки данных могут рассчитывать на организацию сети с большей связностью.

Компания Tandem разработала системы Guardian и NonStop Kernel в расчете на распределенную обработку данных. В течение более двадцати последних лет компания постепенно выявляла и удаляла узкие места в программном обеспечении, ограничивающие возможности масштабирования системы. Чтобы добиться эффективного использования распределенных аппаратных средств при выполнении сложных запросов к базе данных, в базу данных NonStop SQL также были добавлены новые возможности.

В настоящее время в области масштабируемых распределенных вычислений начали широко использоваться также стандартные системы UNIX. В ряде научных приложениях кластеры рабочих станций начали заменять суперкомпьютеры. Предполагается, что эта тенденция станет главной движущей силой для усиленной разработки приложений и операционной среды распределенных вычислений.

В отличие от NonStop Kernel, которая сразу же была разработана как распределенная операционная система, большинство доступных ОС таким свойством не обладают. В качестве механизма масштабирования в этих системах обычно используется симметричная мультипроцессорная обработка (SMP). Этот механизм предполагает реализацию в системе единой разделяемой (общей) памяти, разделяемой системы в/в и когерентности кэш-памяти.

Новая архитектура обеспечивает гибкую реализацию разделяемой памяти. Как показано на рис. 12.8, несколько ЦП могут быть связаны друг с другом с помощью шины когерентности. Вместо использования единой глобальной памяти, память в системе распределена по ЦП. Это приводит к неодинаковому времени доступа к памяти, однако аппаратура поддерживает глобальное адресное пространство и когерентность кэш-памяти.

С целью повышения производительности в каждом ЦП поддерживается свое собственное (локальное) пространство памяти. В каждом процессоре его локальная память начинается с физического адреса 0, ее размер задается в процессе установки конфигурации системы. К локальной памяти может обращаться только локальный процессор.

В такой архитектуре можно выделить три отдельных класса памяти (рис. 12.11). Наименьшее время доступа имеет локальная память, доступ к глобальной памяти, размещенной на плате локального процессора, занимает больше времени, а доступ к глобальной памяти на плате другого ЦП занимает еще больше времени. Локальная память каждого ЦП содержит копию части программного кода и данных операционной системы, доступ к которым разрешен только по чтению. Кроме того, в локальную память операционная система может поместить некоторые частные для процессора данные. Проведенная оценка показала, что размещение этих объектов в локальной памяти и случайное распределение глобальной памяти будет обеспечивать производительность, сравнимую с производительностью системы, реализующей механизмы единой глобальной памяти.

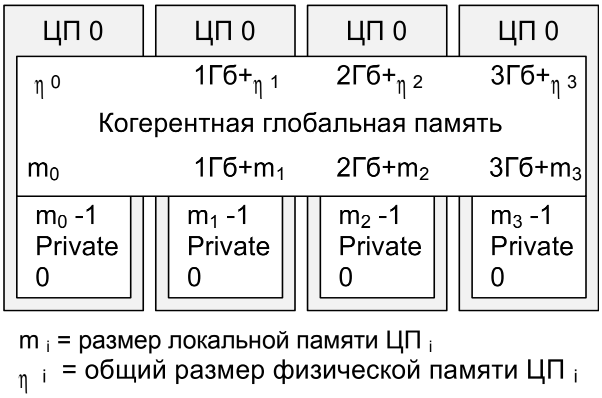


Рис. 12.11. Распределение памяти в четырехпроцессорной SMP-системе

Системы с разделяемой памятью могут функционировать либо в симплексном, либо в дуплексном режиме. Система, построенная на узлах SMP и обеспечивающая отказоустойчивость программными средствами, может использовать симплексный режим. Полностью аппаратная отказоустойчивая система с масштабируемостью SMP требует реализации дуплексного режима работы.

Шина когерентности может также поддерживать работу гибридных распределенных систем, в которых распределенная память используется для обмена данными между процессорами. В этом случае система распределяет большую часть памяти под "локальные нужды" процессоров, а в глобальную память помещает только разделяемые данные.

8.7.4.6. ***12.7.4.6. Система обслуживания***

Основные функции системы обслуживания включают инсталляцию системы, формирование сообщений об ошибках, диагностику и управление средствами контроля питающих напряжений и температурных режимов работы. Системой обслуживания управляют два сервисных процессора (SP), которые размещаются в каждой стойке и работают как специализированные контроллеры в/в ServerNet. SP, размещенные в разных стойках, также связаны друг с другом посредством ServerNet.

Система обслуживания использует специальную систему независимых шин. Эти шины базируются на двух стандартных для промышленности интерфейсах: SPI (Serial Peripheral Interconnect) компании Motorola и систему сканирования в стандарте IEEE 1149.1 JTAG. SPI используется в качестве недорогой последовательной шины в/в для связи со всеми средствами контроля и управления состоянием окружающей среды. Система обслуживания использует средства сканирования для управления, инициализации, тестирования и отображения работы всех СБИС. Применяемое Tandem расширение к стандарту IEEE 1149.1, обеспечивает доступ к регистрам СБИС. Работа средств сканирования никак не затрагивает нормальную работу СБИС. Этот универсальный механизм обеспечивает средство для инициализации СБИС, определения топологии ServerNet и передачи сообщений об ошибках.

8.7.4.7. *12.7.4.7. Инициализация*

Сервисные процессоры несут полную ответственность за инициализацию системы. Во время начальной установки системы они, прежде всего, создают полный перечень доступных в стойке ресурсов, используя шины обслуживания и перепрограммируемые ПЗУ, которые хранят информацию о конфигурации всех компонентов системы. Затем сервисные процессоры инициализируют эти компоненты, используя интерфейс сканирования.

Вслед за этим сервисные процессоры, расположенные в разных стойках, включаются в процесс динамического определения топологии ServerNet. Этот процесс не зависит от ранее полученных данных по конфигурации системы. Он работает ниже уровня передачи пакетов ServerNet, поскольку предшествует установке средств маршрутизации. После того как сервисные процессоры определили топологию сети, они выполняют назначение идентификаторов узлов ServerNet и программирование таблиц маршрутизации в маршрутизаторах.

Затем сервисные процессоры начинают выполнение начальной установки операционной системы на главном процессоре. Сервисные процессоры сообщают программе начальной загрузки и операционной системе перечень аппаратных средств и информацию об их конфигурации, включая адреса ServerNet для каждого устройства системы.

8.7.4.8. *12.7.4.8. Программное обеспечение*

Ключевым свойством новой архитектуры является гибкость. Она полностью поддерживает программно реализованные, распределенные отказоустойчивые системы, подобные NonStop-Kernel. Аппаратные средства позволяют создавать отказоустойчивые SMP-конфигураций для операционных систем, подобных UNIX или Windows NT. Системы могут поддерживать также программные продукты типа Oracle Parallel Server, которые базируются на модели разделяемых общих дисков.

ServerNet позволяет также создавать гибридные системы. Например, одна система может работать под управлением Windows NT в качестве фронтальной машины для базы данных Oracle Parallel Server. При этом системы соединяются друг с другом посредством ServerNet.

Поддержка большого числа операционных систем требует сведения к минимуму усилий по переносу программного обеспечения. С этой целью весь специфический для конкретной платформы код вынесен за рамки операционной системы. При этом для обеспечения отказоустойчивости не требует никаких модификаций в ядре операционной системы. Большая часть этого специфичного для соответствующей платформы кода работает на сервисном процессоре, независимо от операционной системы основных процессоров.

Специальный набор переносимых приложений обеспечивает средства управления системой. Эти приложения отображают состояние аппаратуры, сообщают об ошибках, предпринимают при необходимости автоматические шаги по восстановлению и обеспечивают интерфейсы для удаленного управления системой, используя стандартные для промышленности протоколы.

8.7.5. **12.7.5. Первые системы Tandem на базе технологии ServerNet**

В настоящее время компания Tandem в рамках нового семейства Integrity S4000 выпускает две модели серверов (CM и CO), построенные с использованием технологии ServerNet. Структурная схема одного из базовых вариантов сервера дана на рис. 12.12.

Можно выделить три главные подсистемы: процессорную подсистему, подсистему в/в и подсистему внешней памяти.

Процессорная подсистема строится на базе системных плат (SPU), каждая из которых включает по два микропроцессора с памятью и логикой сравнения, связанные дублированными каналами с подсистемой в/в. В качестве микропроцессоров применяются процессоры MIPS R4400 с кэш-памятью первого уровня емкостью 32 Кбайт (16 Кбайт - кэш команд и 16 Кбайт - кэш данных), работающие на тактовой частоте 200 МГц. Объем кэш-памяти второго уровня составляет 1 Мбайт/процессор. Объем основной памяти системы может достигать 1 Гбайт (в четырехпроцессорной конфигурации).

Подсистема в/в ServerNet создает отказоустойчивую магистраль передачи данных между SPU и контроллерами периферийных устройств и коммуникационными адаптерами. Отказоустойчивость обеспечивается благодаря использованию двух независимых подсетей Servernet. Integrity S4000 поддерживает несколько типов контроллеров в/в. В составе каждого сервера имеется многофункциональный контроллер SSC (Standard System Controller). SSC обеспечивает интерфейс ServerNet с контроллерами в/в, контроллерами SCSI-2 для внутренних устройств массовой памяти, сервисным процессором, а также последовательными и сетевыми интерфейсами для поддержки средств диагностики и консоли. Пара контроллеров SSC обеспечивают отказоустойчивый доступ к устройствам массовой памяти. Каждый контроллер SSC содержит пару интерфейсов шины SCSI-2, которые соединены с другим контроллером SSC и обеспечивают два независимых пути доступа ко всем внутренним дисковым и ленточным накопителям. Система поддерживает "зеркалирование" дисков для обеспечения непрерывного доступа к хранящимся на дисках данным.

В серверы Integrity S4000 могут также устанавливаться дополнительные контроллеры Ethernet, асинхронного и синхронного интерфейсов, каждый из которых имеет по два порта для обеспечения доступа к процессорам через две независимых подсети Servernet. Контроллеры SSC и дополнительные контроллеры в/в могут заменяться в процессе работы системы (в режиме "горячей" замены).

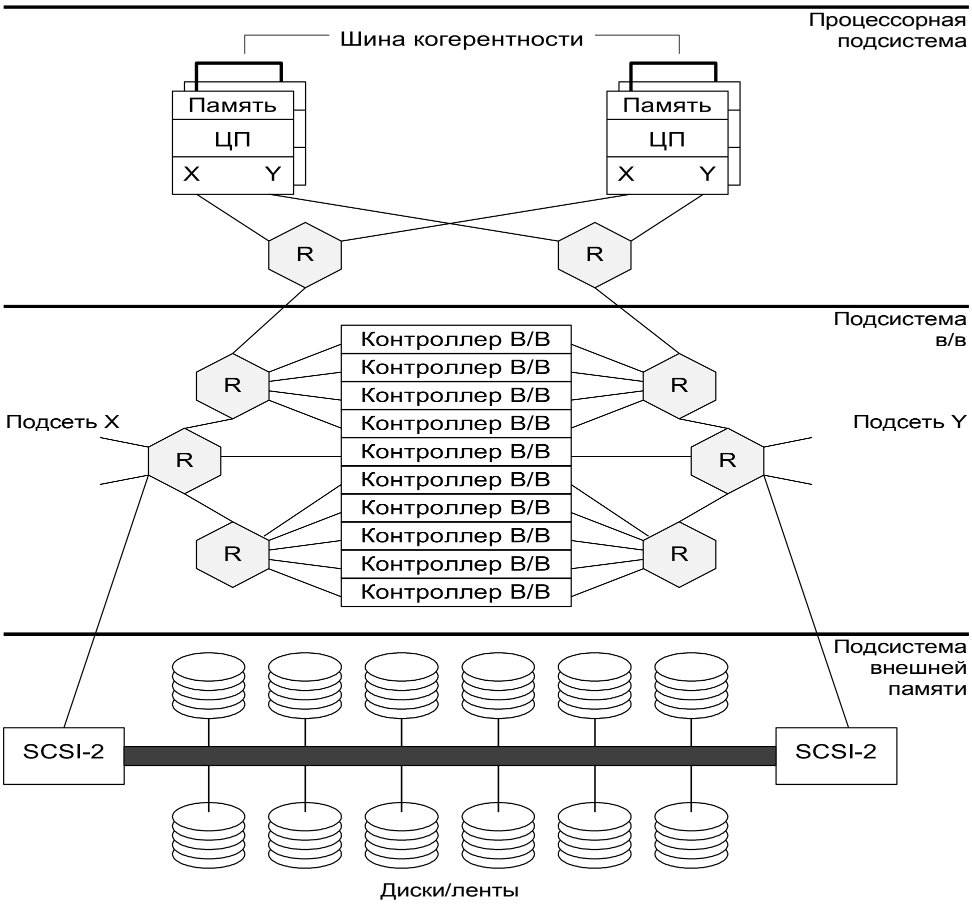


Рис. 12.12. Структурная схема сервера семейства Integrity S4000.

Модели CM и CO отличаются по максимальному количеству устанавливаемых системных плат SPU и характеристиками подсистем в/в (основные параметры приведены в таблице 12.16), а также областью применения. Модели CO специально предназначены для установки в помещениях телекоммуникационных узлов связи и удовлетворяют весьма жестким требованиям (стандарт Bellcore TR-NWT-000063, выпуск 5) по обеспечению повышенной безопасности, устойчивости к пожарам и электрическим характеристикам. Они проверены на устойчивость к землетрясениям, имеют трехуровневую систему аварийной сигнализации и резервные источники питания постоянного тока.

Обе модели работают под управлением операционной системы NonStop-UX, базирующейся на стандарте UNIX System V Release 4.2 MP и обеспечивающей средства повышенной готовности и минимизацию плановых простоев системы, включая средства быстрого обнаружения неисправностей и восстановления, а также возможность модернизации (upgrade) программного обеспечения в режиме online.

Таблица 12.16. Основные параметры моделей CM и CO семейства Integrity S4000

	S4000
Возможности стойки	
Количество плат SPU	4
Процессорные конфигурации: –– Симплексная –– Дуплексная (отказоустойчивая)	 1-4 про 1-2 про
Количество маршрутизаторов	2
Количество плат SSC	2
Количество гнезд в/в ServerNet	1
Количество мест установки устройств внешней памяти	1
Процессор	
Микропроцессор	MIPS RIS
Тактовая частота	200 М
Первичный кэш	16 Кб - к 16 Кб -
Вторичный кэш	1 Мб / п
Основная память	
Объем	128/256Е
Максимально в системе	1024
Пропускная способность шины памяти (пиковая)	400 Мб/
Подсистема в/в	
Количество каналов в/в	2 подсис Serve
Пропускная способность каналов в/в (пиковая)	200 Мб/
Пропускная способность каналов в/в (пиковая)	800 Мб/

8.7.6. **12.7.6. Заключение**

ServerNet создает надежную инфраструктуру для построения и крупномасштабных, и небольших отказоустойчивых систем. Она позволяет объединить между собой несколько сотен процессоров и независимо наращивать количество связей между процессорами и периферийными устройствами, обеспечивая масштабируемую полосу пропускания и избыточные каналы передачи данных. Организация связей типа "каждый с каждым" позволяет предотвратить ненужные промежуточные пересылки данных, которые истощают ресурсы процессоров и памяти. ServerNet предоставляет также средства создания горячего резерва и балансировки нагрузки путем коммутации периферийных устройств между собой.

Базирующиеся на ServerNet системы представляют собой существенный шаг вперед в архитектуре коммерческих отказоустойчивых систем. Одна и та же архитектура дает возможность построения отказоустойчивых систем как чисто аппаратными, так и чисто программными средствами. Она обеспечивает также дополнительную гибкость, поддерживая как модель вычислений в разделяемой общей памяти, так и создание кластеров с распределенной памятью. Из одних и тех же компонентов можно создавать системы разного класса, выбирая операционные системы, процессоры и механизмы поддержки отказоустойчивости.