Национальный исследовательский университет «МЭИ»
Электронный образовательный ресурс

Лекции по курсу
Архитектура компьютера

Дисциплина относится к вариативной части профессионального цикла **Б 2.7** основной образовательной программы подготовки бакалавров по профилю «**Математическое и программное обеспечение вычислительных машин и компьютерных сетей» и «Математическое моделирование»**
Направление подготовки**: 010400 Прикладная математика и информатика**

Автор
Шамаева О.Ю.

Москва, 2015

Оглавление

Лекция 1. Основные концепции и тенденции развития архитектур современных компьютеров

Основные вопросы:

1.1. Введение
1.2. Классификации ЭВМ и ВС по М. Флинну и Р. Хокни
1.3. Особенности организации и функционирования архитектур с общей, распределенной и смешанной памятью
1.3.1. Массивно-параллельные системы (MPP)
1.3.2. Симметричные мультипроцессорные системы (SMP)
1.3.3. Системы с неоднородным доступом к памяти (NUMA)
1.3.4. Параллельные векторные системы (PVP)
1.3.5. Кластерные системы
1.4. Организация схем коммутаций
1.4.1. Организация схем коммутации в МВС с общей памятью
1.4.2. Организация средств коммутации в архитектуре “Butterfly”
1.4.3. Организация схем коммутации в МВС с распределенной памятью
1.4.4. Архитектура систем со смешанной организацией памяти

1.1. Введение

Стремительное развитие науки и проникновение человеческой мысли во все новые области вместе с решением поставленных прежде проблем постоянно порождает поток вопросов и ставит новые, как правило, более сложные задачи. Во времена первых компьютеров казалось, что увеличение их быстродействия в 100 раз позволит решить большинство проблем, однако гигафлопная производительность современных суперЭВМ сегодня является явно недостаточной для многих ученых.

Электро- и гидродинамика, сейсморазведка, прогноз погоды, моделирование химических соединений, исследование виртуальной реальности – вот далеко не полный список областей науки, исследователи которых используют каждую возможность ускорить выполнение своих программ.

Наиболее перспективным и динамичным направлением увеличения скорости решения прикладных задач ***является широкое внедрение идей параллелизма в работу вычислительных систем***. В научной литературе и технической документации можно найти более десятка различных названий, характеризующих лишь общие принципы функционирования параллельных машин: векторно-конвейерные, массивно-параллельные, компьютеры с широким командным словом, систолические массивы, гиперкубы, спецпроцессоры и мультипроцессоры, иерархические и кластерные компьютеры, dataflow, матричные ЭВМ и многие другие. Если же к подобным названиям для полноты описания добавить еще и данные о таких важных параметрах, как, например, организация памяти, топология связи между процессорами, синхронность работы отдельных устройств или способ исполнения арифметических операций, то число различных архитектур станет и вовсе необозримым.

Попытки систематизировать все множество архитектур начались после опубликования в конце 60-х годов **М.Флинном** первого варианта классификации вычислительных систем, и непрерывно продолжаются по сей день. Ясно, что навести порядок в хаосе очень важно для лучшего понимания исследуемой предметной области, однако нахождение удачной классификации может иметь целый ряд существенных следствий.

В самом деле, вспомним открытый в 1869 году **Д.И.Менделеевым** периодический закон. Выписав на карточках названия химических элементов и указав их важнейшие свойства, он сумел найти такое расположение, при котором четко прослеживалась закономерность в изменении свойств элементов, расположенных в каждом столбце и в каждой строке. Теперь, зная положение какого-либо элемента в таблице, он мог с большой степенью точности описывать его свойства, не проводя с ним никаких непосредственных экспериментов. Другим, поистине фантастическим следствием, явилось то, что данный закон сразу указал на несколько "белых пятен" в таблице и позволил предсказать существование и свойства неизвестных до тех пор элементов. В 1875 году французский ученый Буабодран, изучая спектры минералов, открыл предсказанный Менделеевым галлий и впервые подробно описал его свойства. В свою очередь Менделеев, никогда прежде не видевший данного химического элемента, не только смог указать на ошибку в определении плотности, но и вычислил ее правильное значение.

Существующая классификация растительного и животного мира, в отличие от периодического закона, носит скорее описательный характер. С ее помощью намного сложнее предсказывать существование нового вида, однако знание того, что исследуемый экземпляр принадлежит к тому или иному роду/семейству/отряду/классу позволяет оправданно предположить наличие у него вполне определенных свойств.

Подобную классификацию хотелось бы найти и для архитектур параллельных вычислительных систем. Основной вопрос – что заложить в основу классификации – может решаться по-разному, в зависимости от того, для кого данная классификация создается и на решение какой задачи направлена.

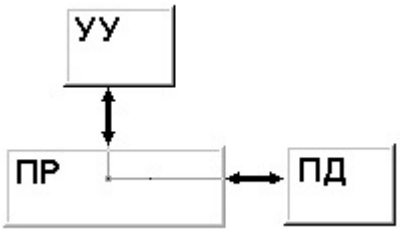
Удачная классификация могла бы подсказать возможные пути совершенствования компьютеров и в этом смысле она должна быть достаточно содержательной. Трудно рассчитывать на нахождение нетривиальных «белых пятен», но размышления о возможной систематике с точки зрения простоты и технологичности программирования могут оказаться чрезвычайно полезными для определения направлений поиска новых архитектур.

Рассмотрим наиболее известные на сегодня классификации.

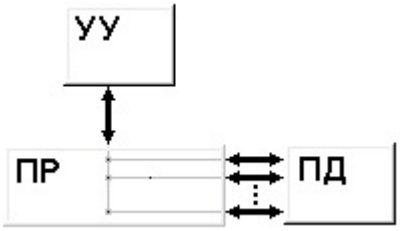
1.2. Классификации ЭВМ и ВС по М. Флинну и Р. Хокни

По-видимому, самой ранней и наиболее известной является классификация архитектур вычислительных систем, предложенная в 1966 году **М. Флинном**. Классификация базируется на понятии ***потока,*** под которым понимается **последовательность элементов, команд или данных, обрабатываемая**

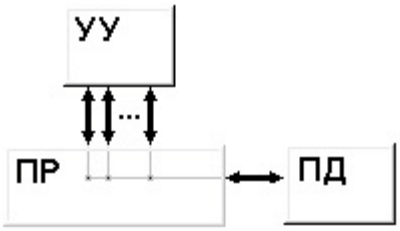
процессором. На основе числа потоков команд и потоков данных М. Флинн выделяет четыре класса архитектур: **SISD, MISD, SIMD и MIMD**.



SISD (single instruction stream/single data stream) – одиночный поток команд и одиночный поток данных. К этому классу относятся, прежде всего, классические последовательные машины, или иначе, машины фон-неймановского типа, например, PDP-11 или VAX 11/780. В таких машинах есть только один поток команд, все команды обрабатываются последовательно друг за другом и каждая команда инициирует одну операцию с одним потоком данных. Для увеличения скорости обработки команд и скорости выполнения арифметических операций может применяться конвейерная обработка. Поэтому, как машина CDC 6600 со скалярными функциональными устройствами, так и CDC 7600 с конвейерными попадают в этот класс.



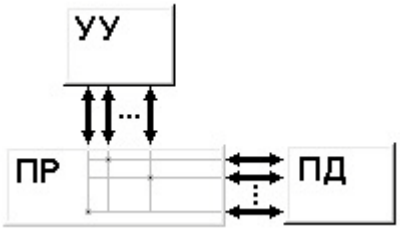
SIMD (single instruction stream/multiple data stream) – одиночный поток команд и множественный поток данных. В архитектурах подобного рода сохраняется один поток команд, включающий, в отличие от предыдущего класса, векторные команды. Это позволяет выполнять одну арифметическую операцию сразу над многими данными – элементами вектора. Способ выполнения векторных операций не оговаривается, поэтому обработка элементов вектора может производиться либо процессорной матрицей, как в ILLIAC IV, либо с помощью конвейера, как, например, в машине CRAY-1.



MISD (multiple instruction stream/single data stream) – множественный поток команд и одиночный поток данных. Определение подразумевает наличие в архитектуре многих процессоров, обрабатывающих один и тот же поток данных. Однако ни М.Флинн, ни другие специалисты в области архитектуры компьютеров до некоторого времени не могли представить убедительный пример реально существующей вычислительной системы, построенной на данном принципе. Ряд исследователей относят конвейерные машины к данному классу, однако это не нашло

окончательного признания в научном сообществе.

Появившиеся многоядерные компьютеры можно отнести к данному классу.



MIMD (multiple instruction stream/multiple data stream) – множественный поток команд и множественный поток данных. Этот класс предполагает, что в вычислительной системе есть несколько устройств обработки команд, объединенных в единый комплекс и работающих каждое со своим потоком команд и данных.

Можно отметить два недостатка в классификации **М. Флинна**. Во-первых, некоторые заслуживающие внимания архитектуры, например, dataflow и векторно-конвейерные машины, четко не вписываются в данную классификацию. Во-вторых, класс MIMD чрезвычайно заполнен. Поэтому необходимо средство, более избирательно систематизирующее архитектуры, которые по М.Флинну попадают в один класс, но совершенно различны по числу процессоров, природе и топологии связи между ними, по способу организации памяти и, конечно же, по технологии программирования.

Роджер Хокни разработал свой подход к классификации для более детальной систематизации компьютеров, попадающих в класс MIMD по **систематике М. Флинна**. Пытаясь систематизировать архитектуры внутри этого класса, Р. Хокни получил иерархическую структуру, представленную на рис. 1.1.

Рис. 1.1. Иерархическая структура архитектуры класса MIMD

Основная идея классификации состоит в следующем. Множественный поток команд может быть обработан двумя способами: либо одним конвейерным устройством обработки, работающем в режиме разделения времени для отдельных потоков, либо каждый поток обрабатывается своим собственным устройством. Первая возможность используется в MIMD-компьютерах, получивших название конвейерных. Архитектуры, использующие вторую возможность, в свою очередь, делятся на два класса. В первый класс попадают MIMD-компьютеры, в которых возможна прямая связь каждого процессора с каждым, реализуемая с помощью переключателя. Во втором классе находятся MIMD-компьютеры, в которых прямая связь каждого процессора возможна только с ближайшими соседями по сети, а взаимодействие удаленных процессоров поддерживается специальной системой маршрутизации.

Среди MIMD-машин с переключателем Р. Хокни выделяет те, в которых вся память распределена среди процессов как их локальная память, например, PASM, PRINGLE, IBM SP2 без SMP-узлов. В этом случае общение самих процессоров реализуется с помощью сложного переключателя, составляющего значительную часть компьютера. Такие машины носят название ***MIMD-машин с распределенной памятью***. Если память – разделяемый ресурс, доступный всем процессорам через переключатель, то MIMD-машины являются системами с общей памятью (BBN Butterfly, Cray C90). В соответствии с типом переключателей можно проводить классификацию и далее: простой переключатель, многокаскадный переключатель, общая шина и т. п. Многие современные вычислительные системы имеют как общую разделяемую память, так и распределенную локальную. Такие системы принято называть гибридными MIMD с переключателем.

1.3. Особенности организации и функционирования архитектур с общей, распределенной и смешанной памятью

Классифицируя современные компьютеры, которые практически все относятся к классу MIMD, будем основываться на анализе используемых в системах способах организации оперативной памяти. На рис. 1.2 приведена классификация систем MIMD, основанная на разных способах организации памяти.

Данный поход позволяет различать два важных типа многопроцессорных систем – **мультипроцессоры** (multiprocessors или системы с общей разделяемой памятью) и **мультикомпьютеры** (multicomputers или системы с распределенной памятью).

Для **мультипроцессоров** учитывается способ построения общей памяти. Возможный подход – использование единой (централизованной) общей памяти. Такой подход обеспечивает однородный доступ к памяти (uniform memory access or UMA) и служит основой для построения векторных суперкомпьютеров (parallel vector processor, PVP) и симметричных мультипроцессоров (symmetric multiprocessor or SMP). Среди примеров первой группы суперкомпьютер Cray T90, ко второй группе относятся IBM eServer p690, Sun Fire E15K, HP Superdome, SGI Origin 300 и др.

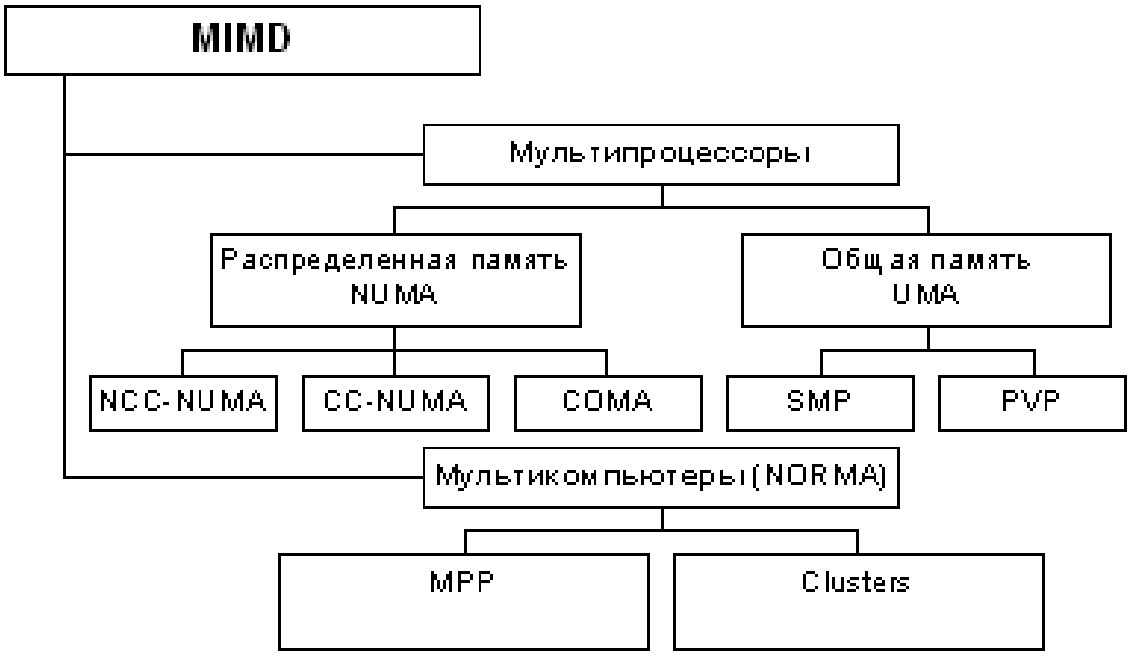


Рис. 1.2. Классификация систем MIMD

Общий доступ к данным может быть обеспечен и при физически распределенной памяти (при этом, естественно, длительность доступа уже не будет одинаковой для всех элементов памяти). Такой подход именуется как неоднородный доступ к памяти (non-uniform memory access or NUMA).

Среди систем с таким типом памяти выделяют:

- Системы, в которых для представления данных используется только локальная кэш память имеющихся процессоров (cache-only memory architecture or COMA); примерами таких систем являются, например, KSR-1 и DDM;
- Системы, в которых обеспечивается однозначность (когерентность) локальных кэш памяти разных процессоров (cache-coherent NUMA or CC-NUMA); среди систем данного типа SGI Origin2000, Sun HPC 10000, IBM/Sequent NUMA-Q 2000;
- Системы, в которых обеспечивается общий доступ к локальной памяти разных процессоров без поддержки на аппаратном уровне когерентности кэша (non-cache coherent NUMA or NCC-NUMA); к данному типу относится, например, система Cray T3E.

Мультикомпьютеры уже не обеспечивают общий доступ ко всей имеющейся в системах памяти (no-remote memory access or NORMA). Данный подход используется при построении двух важных типов многопроцессорных вычислительных систем – массивно-параллельных систем (massively parallel processor or MPP) и кластеров (clusters). Среди представителей первого типа систем – IBM RS/6000 SP2, Intel PARAGON/ASCI Red, транспьютерные системы Parsytec и др.; примерами кластеров являются, например, системы AC3 Velocity, NCSA/NT Supercluster и др.

Следует отметить чрезвычайно быстрое развитие кластерного типа многопроцессорных вычислительных систем.

Далее приводятся базовые характеристики основных классов современных компьютеров.

1.3.1. Массивно-параллельные системы (MPP)

На рис. 1.3 представлена типовая архитектура вычислительных систем с распределенной памятью.

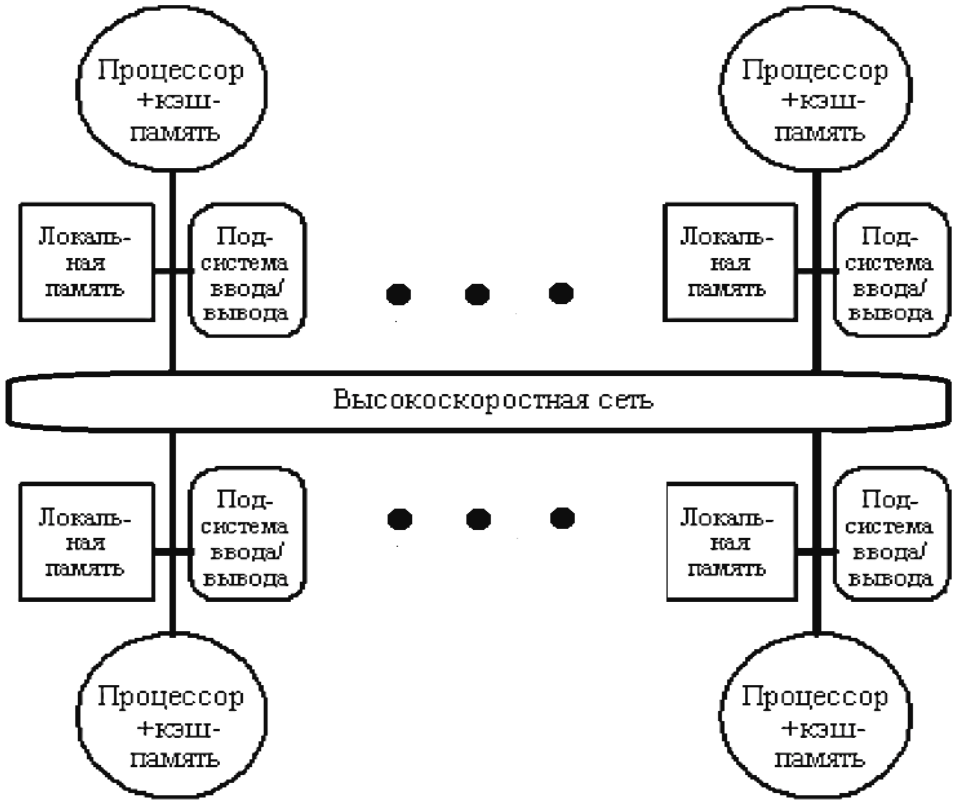


Рис. 1.3. Типовая архитектура машины с распределенной памятью

Архитектура: Система состоит из однородных вычислительных узлов, включающих:

- один или несколько центральных процессоров (обычно RISC),
- локальную память (прямой доступ к памяти других узлов невозможен),
- коммуникационный процессор или сетевой адаптер,
- иногда – жесткие диски (как в SP) и/или другие устройства Вв/Выв.

К системе могут быть добавлены специальные узлы ввода-вывода и управляющие узлы. Узлы связаны через некоторую коммуникационную среду (высокоскоростная сеть, коммутатор и т.п.)

Примеры: IBM RS/6000 SP2, Intel PARAGON/ASCI Red, SGI/CRAY T3E, Hitachi SR8000, транспьютерные системы Parsytec и др.

Масштабируемость: Общее число процессоров в реальных системах достигает нескольких тысяч (ASCI Red, Blue Mountain).

Операционная система: Существуют два основных варианта реализации:

1. полноценная ОС работает только на управляющей машине (front-end), на каждом узле работает сильно урезанный вариант ОС,

обеспечивающий только работу расположенной в нем ветви параллельного приложения. Пример: Cray T3E.

2. на каждом узле работает полноценная UNIX-подобная ОС (вариант, близкий к кластерному подходу).

Пример: IBM RS/6000 SP + операционная система AIX, которая устанавливается отдельно на каждом узле.

Модель программирования: Программирование в рамках модели передачи сообщений (MPI, PVM, BSPlib)

Распределенность памяти означает, что каждый процессор имеет непосредственный доступ только к своей локальной памяти, а доступ к данным, расположенным в памяти других процессоров, выполняется другими способами.

Чтобы переслать информацию от процессора к процессору, необходим механизм передачи сообщений по сети, связывающей вычислительные узлы. Для абстрагирования от подробностей функционирования коммуникационной аппаратуры и программирования на высоком уровне, используются библиотеки передачи сообщений. Несмотря на существенные различия средств межпроцессорного взаимодействия в разных системах по скоростным параметрам и по способу аппаратной реализации, библиотеки обмена сообщениями выполняют приблизительно одни и те же функции.

Выбор топологии машины часто определяет способ решения прикладной задачи. Надо заметить, что оптимизация алгоритмов для параллельных архитектур существенно отличается от той же работы для последовательных систем. Если переход с одного скалярного процессора на другой практически никогда не требует пересмотра алгоритма, то алгоритм, идеально приспособленный для одной параллельной архитектуры, на другой машине (с тем же числом процессоров того же типа) может работать неприемлемо медленно. Для оценки производительности распределенной системы, кроме топологии связей, необходимо знать скорость выполнения арифметических операций, время инициализации канала связи и время передачи единицы объема информации. Если топология системы не тривиальна, то в состав операционной системы или пакета передачи сообщений приходится включать процедуры маршрутизации сообщений, работающие на каждом узле и обеспечивающие пересылку транзитных сообщений. Они также вызывают задержку при передаче информации между узлами, не имеющими прямого канала связи.

Таким образом, применение дешевых процессоров позволяет сделать относительно недорогой суперкомпьютер. Широкому распространению подобных архитектур препятствует в основном отсутствие эффективных параллельных программ, полностью использующих их возможности.

1.3.2. Симметричные мультипроцессорные системы (SMP)

На рис. 1.4 приведена типовая архитектура мультипроцессорных вычислительных систем с общей памятью.

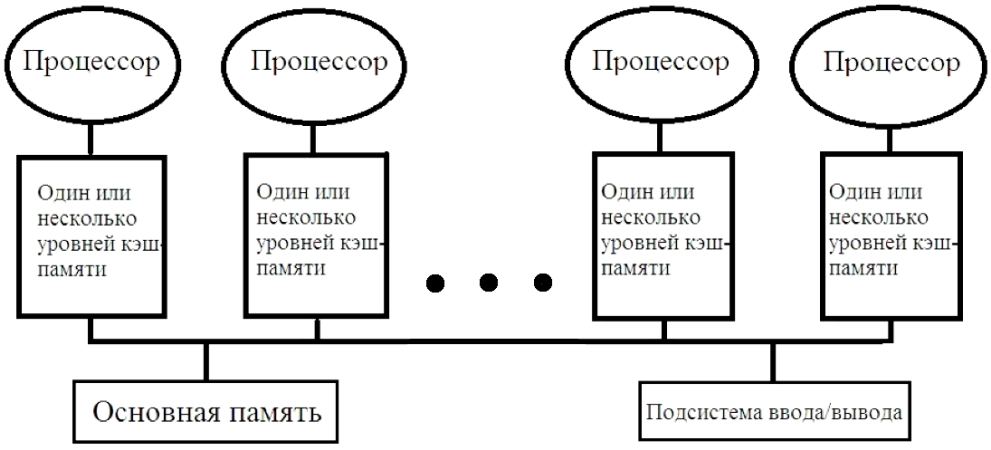


Рис. 1.4. Типовая архитектура мультипроцессорной системы с общей памятью

Архитектура: Система состоит из нескольких однородных процессоров и массива общей памяти (обычно из нескольких независимых блоков). Все процессоры имеют доступ к любой точке памяти с одинаковой скоростью. Процессоры подключены к памяти либо с помощью общей шины (базовые 2–4 процессорные SMP-сервера), либо с помощью crossbar-коммутатора (HP 9000). Аппаратно поддерживается когерентность кэшей.

Примеры: HP 9000 V-class, N-class; SMP-сервера и рабочие станции на базе процессоров Intel (IBM, HP, Compaq, Dell, ALR, Unisys, DG, Fujitsu и др.).

Масштабируемость: Наличие общей памяти сильно упрощает взаимодействие процессоров между собой, однако накладывает сильные ограничения на их число – не более 32 в реальных системах. Для построения масштабируемых систем на базе SMP используются кластерные или NUMA-архитектуры.

Операционная система: Вся система работает под управлением единой ОС (обычно UNIX-подобной, но для Intel-платформ поддерживается Windows NT). ОС автоматически (в процессе работы) распределяет процессы/нити по процессорам (scheduling), но иногда возможна и явная привязка.

Модель программирования: Программирование в модели общей памяти. (POSIX threads, OpenMP). Для SMP-систем существуют сравнительно эффективные средства автоматического распараллеливания.

SMP - это один компьютер с несколькими равноправными процессорами, но с одной памятью, подсистемой ввода/вывода и одной ОС. Каждый процессор имеет доступ ко всей памяти, может выполнять любую операцию ввода/вывода, прерывать другие процессоры и т.д., но это представление справедливо только на уровне программного обеспечения. На самом же деле в SMP имеется несколько устройств памяти.

Каждый процессор имеет, по крайней мере, одну собственную кэш-память, что необходимо для достижения хорошей производительности, поскольку основная память работает слишком медленно по сравнению со скоростью процессоров (и это соотношение все больше ухудшается), а кэш работает со скоростью процессора, но дорог, и поэтому устройства кэш-памяти обладают относительно небольшой емкостью. Из-за этого в кэш помещается лишь оперативная информация, остальное же хранится в основной памяти. Отсюда возникает проблема когерентности кэшей – получение процессором значения, находящегося в кэш-память другого процессора. Это решается при помощи отправки широковещательного запроса всем устройствам кэш-памяти, основной памяти и даже подсистеме ввода/вывода, если она работает с основной памятью напрямую, с целью получения актуальной информации.

Имеется еще одно следствие, связанное с параллелизмом. Неявно производимая аппаратурой SMP пересылка данных между кэшами является наиболее быстрым и самым дешевым средством коммуникации в любой параллельной архитектуре общего назначения. Поэтому при наличии большого числа коротких транзакций (свойственных, например, банковским приложениям), когда приходится часто синхронизовать доступ к общим данным, архитектура SMP является наилучшим выбором; любая другая архитектура работает хуже.

Тем не менее, архитектуры с разделяемой общей памятью не считаются перспективными. Основная причина довольно проста. Рост производительности в параллельных системах обеспечивается наращиванием числа процессоров, что приводит к тому, что узким местом становится доступ к памяти. Увеличение локальной кэш-памяти не способно полностью решить проблему: задача поддержания согласованного состояния нескольких банков кэш-памяти столь же трудна. Как правило, на основе общей памяти не создают систем с числом процессоров более 32, при необходимости объединяя их в кластерные или NUMA-архитектуры.

1.3.3. Системы с неоднородным доступом к памяти (NUMA)

На рис. 1.5 изображена типовая архитектура мультипроцессорных вычислительных систем с неоднородным доступом к памяти.

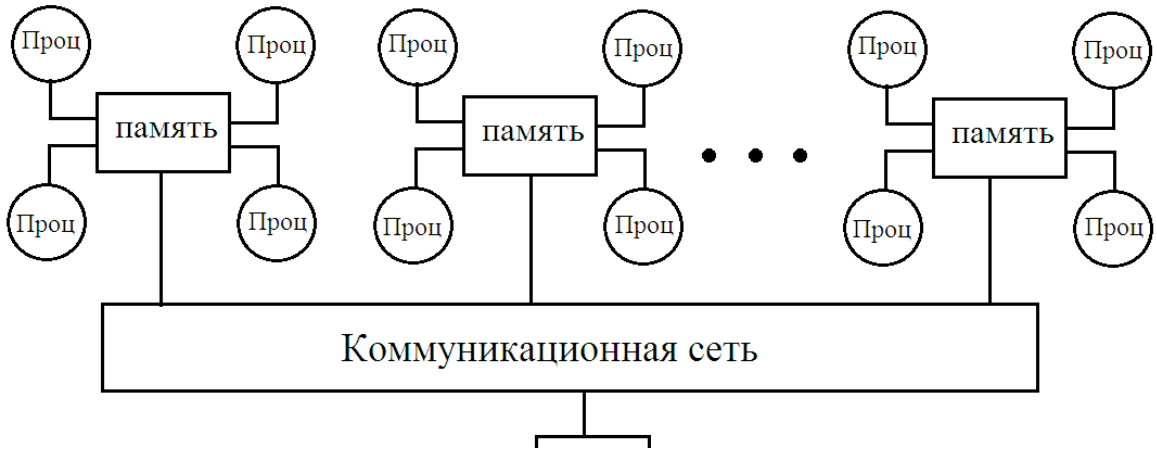


Рис. 1.5. Типовая архитектура мультипроцессорной системы с неоднородным доступом к памяти

Архитектура: Система состоит из однородных базовых модулей (плат), состоящих из небольшого числа процессоров и блока памяти. Модули объединены с помощью высокоскоростного коммутатора. Поддерживается единое адресное пространство, аппаратно поддерживается доступ к удаленной памяти, т.е. к памяти других модулей. Доступ к локальной памяти узла осуществляется в несколько раз быстрее, чем к удаленной.

В случае, если аппаратно поддерживается когерентность кэшей во всей системе (обычно это так), говорят об архитектуре cc-NUMA (cache-coherent NUMA)

Примеры: HP 9000 V-class в SCA-конфигурациях, SGI Origin2000, Sun HPC 10000, IBM/Sequent NUMA-Q 2000, SNI RM600 и др.

Масштабируемость: Масштабируемость NUMA-систем ограничивается объемом адресного пространства, возможностями аппаратуры поддержки когерентности кэшей и возможностями операционной системы по управлению большим числом процессоров. На настоящий момент, максимальное число процессоров в NUMA-системах составляет 256 (Origin2000).

Операционная система: Обычно вся система работает под управлением единой ОС, как в SMP. Но возможны также варианты динамического "подразделения" системы, когда отдельные "разделы" системы работают под управлением разных ОС (например, Windows NT и UNIX в NUMA-Q 2000).

Модель программирования: Аналогично SMP.

По сути своей NUMA представляет собой большую SMP-систему, разбитую на набор более мелких и простых SMP. Аппаратура позволяет работать со всеми отдельными устройствами основной памяти составных частей системы (называемых обычно узлами) как с единой гигантской памятью. Этот подход порождает ряд следствий. Во-первых, в системе имеется одно адресное пространство, распространяемое на все узлы. Реальный (не виртуальный) адрес 0 для каждого процессора в любом узле соответствует адресу 0 в частной памяти узла 0; реальный адрес 1 для всей машины – это адрес 1 в узле 0 и т.д., пока не будет использована вся память узла 0. Затем происходит переход к памяти узла 1, затем узла 2 и т.д. Для реализации этого единого адресного пространства каждый узел NUMA включает специальную аппаратуру (Dir), которая решает проблему когерентности кэшей, обеспечивая получение актуальной информации от других узлов.

Понятно, что этот процесс длится несколько дольше, чем если бы требуемое значение находилось в частной памяти того же узла. Отсюда и происходит словосочетание "неоднородный доступ к памяти". В отличие от SMP, время выборки значения зависит от адреса и от того, от какого процессора исходит запрос (если, конечно, требуемое значение не содержится в кэше).

Поэтому ключевым вопросом является степень "неоднородности" NUMA. Например, если для взятия значения из другого узла требуется только на 10% большее время, то это никого не задевает. В этом случае все будут относиться к системе как к SMP, и разработанные для SMP программы будут выполняться достаточно хорошо.

Однако в текущем поколении NUMA-систем для соединения узлов используется сеть. Это позволяет включать в систему большее число узлов, до 64 узлов с общим числом процессоров 128 в некоторых системах. В результате, современные NUMA-системы не выдерживают правила 10% – лучшие образцы замедление 200–300% и даже более. При такой разнице в скорости доступа к памяти для обеспечения должной эффективности следует позаботиться о правильном расположении требуемых данных. Чтобы этого добиться, можно соответствующим образом модифицировать операционную систему (и это сделали поставщики систем в архитектуре NUMA). Например, такая операционная система при запросе из программы блока памяти выделяет память в узле, в котором выполняется эта программа, так что когда процессор ищет соответствующие данные, то находит их в своем собственном узле. Аналогичным образом должны быть изменены подсистемы (включая СУБД), осуществляющие собственное планирование и распределение памяти (что и сделали Oracle и Informix). Как утверждает компания Silicon Graphics, такие изменения позволяют эффективно выполнять в системах с архитектурой NUMA приложения, разработанные для SMP, без потребности изменения кода.

1.3.4. Параллельные векторные системы (PVP)

Архитектура: Основным признаком PVP-систем является наличие специальных векторно-конвейерных процессоров, в которых предусмотрены команды однотипной обработки векторов независимых данных, эффективно выполняющиеся на конвейерных функциональных устройствах.

Как правило, несколько таких процессоров (1–16) работают одновременно над общей памятью (аналогично SMP) в рамках многопроцессорных конфигураций. Несколько таких узлов могут быть объединены с помощью коммутатора (аналогично MPP).

Примеры: NEC SX-4/SX-5, линия векторно-конвейерных компьютеров CRAY: от CRAY-1, CRAY J90/T90, CRAY SV1, серия Fujitsu VPP и др.

Модель программирования: Эффективное программирование подразумевает векторизацию циклов (для достижения разумной производительности одного процессора) и их распараллеливание (для одновременной загрузки нескольких процессоров одним приложением).

1.3.5. Кластерные системы

Архитектура: Набор рабочих станций (или даже ПК) общего назначения, используется в качестве дешевого варианта массивно-параллельного компьютера. Для связи узлов используется одна из стандартных сетевых технологий (Fast/Gigabit Ethernet, Myrinet и др.) на базе шинной архитектуры или коммутатора.

При объединении в кластер компьютеров разной мощности или разной архитектуры, говорят о гетерогенных (неоднородных) кластерах.

Узлы кластера могут одновременно использоваться в качестве пользовательских рабочих станций. В случае, когда это не нужно, узлы могут быть существенно облегчены и/или установлены в стойку.

Примеры: NT-кластер в NCSA, Beowulf-кластеры, кластеры МГУ и СПбГУ, кластер МЭИ (см.рис. 1.6 и 1.7) и др.

Операционная система: Используются стандартные для рабочих станций ОС, чаще всего, свободно распространяемые – Linux/FreeBSD, вместе со специальными средствами поддержки параллельного программирования и распределения нагрузки.

Модель программирования: Программирование, как правило, в рамках модели передачи сообщений (чаще всего - MPI).

Кластер МЭИ (ТУ)

В 2007 г. в Московском Энергетическом Институте был установлен кластер с пиковой производительностью 281 *Flops*. Производительность кластера МЭИ при тестировании на *LinPack* составляет 230 *flops.*

Все узлы кластера реализованы на базе серверов SUN Fire X4100. В его состав входят 16 вычислительных узлов (computing node – CN) (см. рис. 1.6.) и один управляющий узел (control node – Con N) (см. рис.1.7). Каждый из CN выполнен на двух двухядерных процессорах AMD Opteron 275, а Con N на одноядерных процессорах AMD Opteron 254 .

Каждый CN имеет в своем составе общую память емкостью 4ГБ (MEM 1 и MEM 2) и дисковую память HDD емкостью 2*73 ГБ. Con N имеет общую память емкостью 8ГБ и дисковую – 73ГБ. Связь между ядрами CN осуществляется с помощью локальной сети AMD Hyper Transport, связь между узлами с помощью системной сети InfiniBand, связь с внешним миром осуществляется с помощью вспомогательной сети Gigabit Ethernet. В качестве операционной системы выступает ОС SUSE LINUX Enterprise Server 10

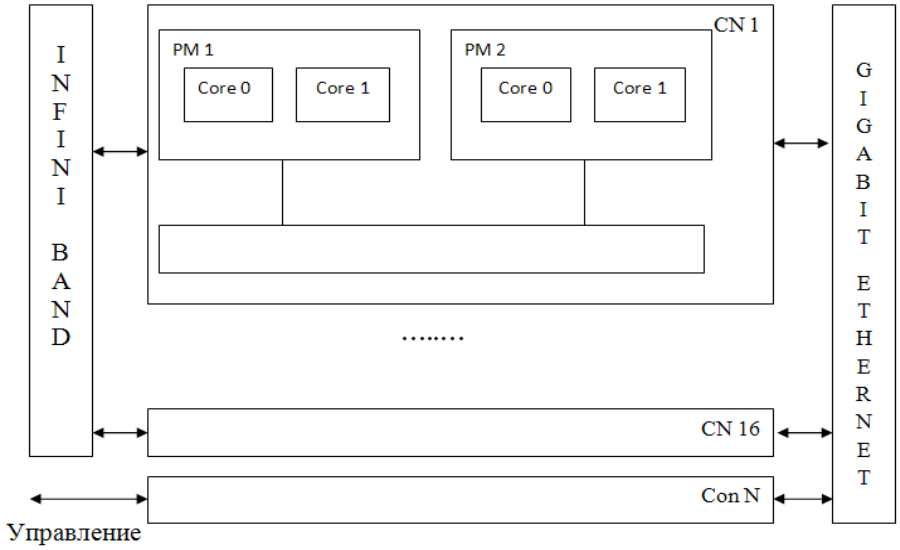
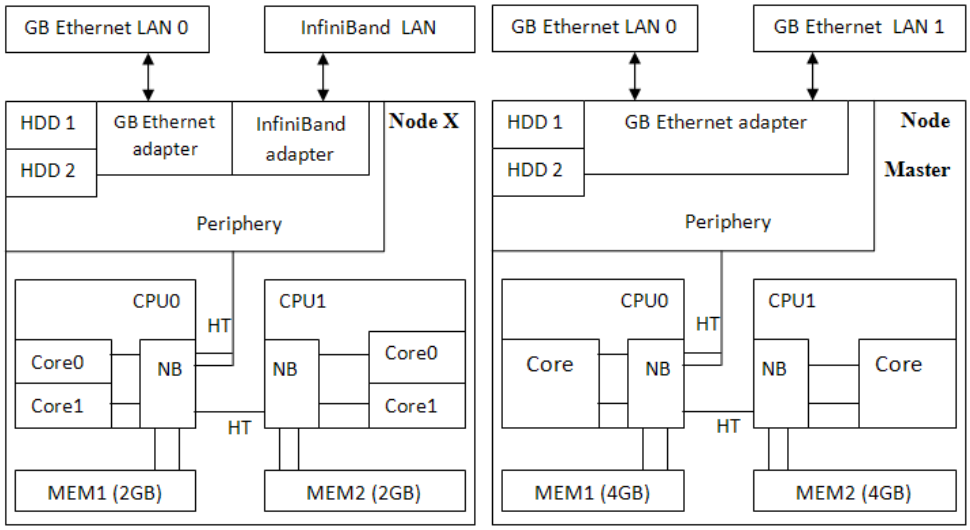


Рис. 1.6. Вычислительный кластер МЭИ



а) б)

Рис. 1.7. Структурные схемы вычислительного (а) и управляющего (б) узлов кластера

Каждый процессор имеет доступ только к одному своему блоку оперативной памяти. Доступ к другому блоку памяти возможен только через контроллер памяти другого процессора.

При построении кластера МЭИ использовались 3 технологии передачи данных: InfiniBand, HyperTransport, Gigabit Ethernet.

1.4. Организация схем коммутации

Важнейшим аспектом создания высокопроизводительных архитектур является построение средств коммутации.

Структура линий коммутации между процессорами вычислительной системы (топология сети передачи данных) определяется, как правило, с учетом возможностей технической реализации. Немаловажную роль при выборе структуры сети играет и анализ интенсивности информационных потоков при параллельном решении наиболее распространенных вычислительных задач. Рассмотрим основные схемы коммутации в МВС разной системы организации памяти.

1.4.1. Организация схем коммутации в МВС с общей памятью

На рис. 1.8 представлена типовая архитектура МВС с общей памятью.

Замечание. В реальных вычислительных системах (ВС) с общей памятью количество процессоров n не превосходит 32. Число модулей памяти m, осуществляющих хранение данных, не превосходит n, то есть $m \le n \le 32$.

Возможно несколько вариантов построения ВС с общей памятью:

1. ВС с единственным модулем памяти ($m = 1$), или многовходовой памятью;
2. ВС с несколькими модулями памяти ($1 < m < n$), или несколькими многовходовыми памятями;
3. ВС с количеством модулей памяти равным количеству процессоров ($m = n$) и коммутацией в виде коммутационной сети.

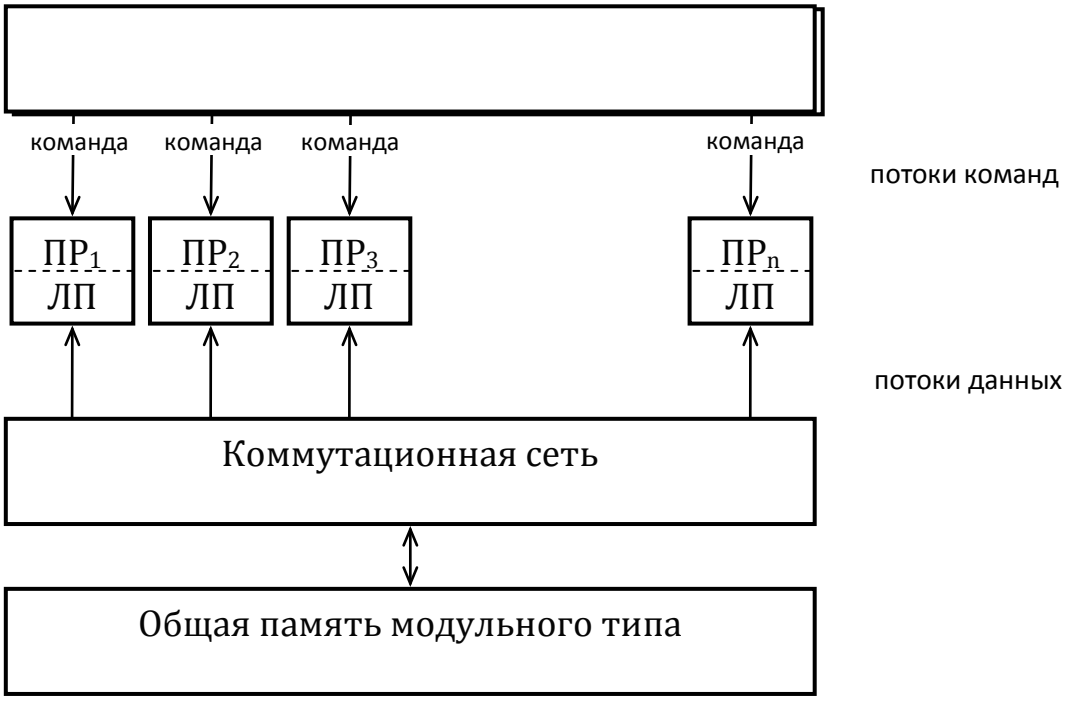


Рис. 1.8. Типовая архитектура МВС с общей памятью

На рис. 1.9, 1.10 и 1.11 представлены варианты реализации.

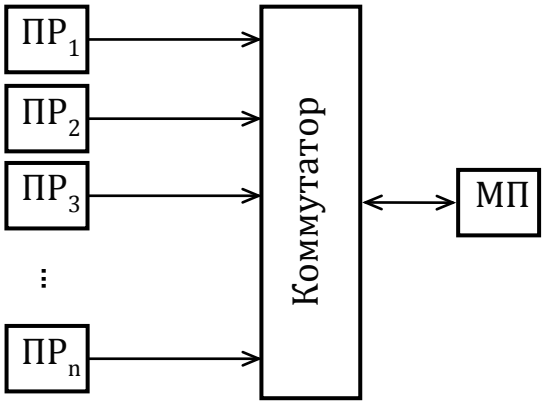


Рис. 1.9. Единый модуль памяти $m = 1$

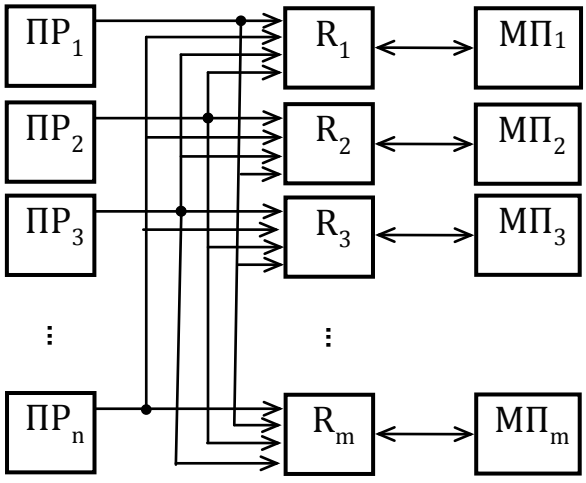


Рис. 1.10. Несколько модулей памяти $1 < m < n$

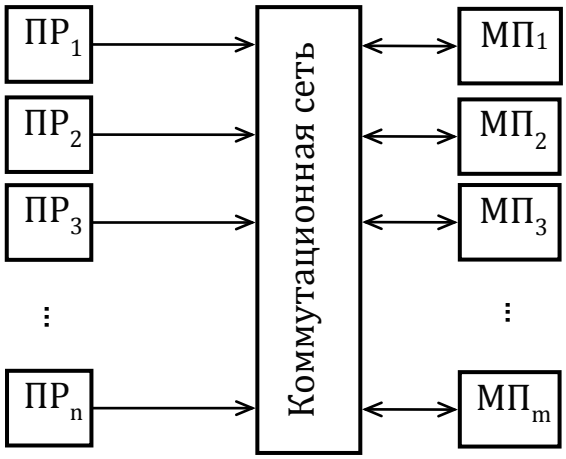


Рис. 1.11. Число модулей памяти равно числу процессоров $m = n$

1.4.2. Организация средств коммутации в архитектуре “Butterfly”

Для архитектуры “Butterfly” - $n = m = 256$. Коммутационный узел (КУ), как изображено на рис. 1.12, имеет 4 входа и 4 выхода, причем каждый вход соединён с каждым выходом.

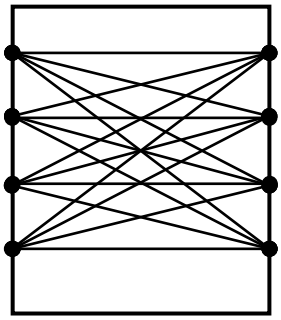


Рис. 1.12. Коммутационный узел «4 х 4»

Коммутационный блок (КБ) представляет собой аналогичную КУ структуру, «атомарными» в которой являются КУ (рис. 1.13).

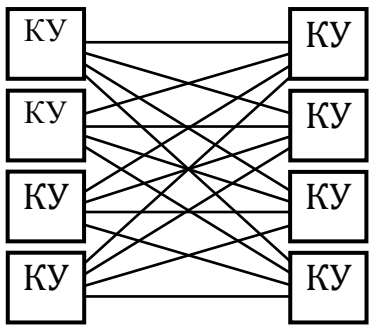


Рис. 1.13. Коммутационный блок «16 х 16»

Коммутационная сеть по типу Butterfly изображена на рис. 1.14.

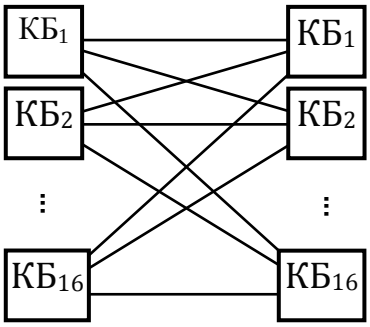


Рис. 1.14. Коммутационная сеть по типу Butterfly «256 х 256»

В МВС с общей памятью в качестве средства передачи информации часто используют шины. Каждый модуль памяти имеет свою шину, к которой подключаются все процессорные узлы. Схематически это показано на рис. 1.15.

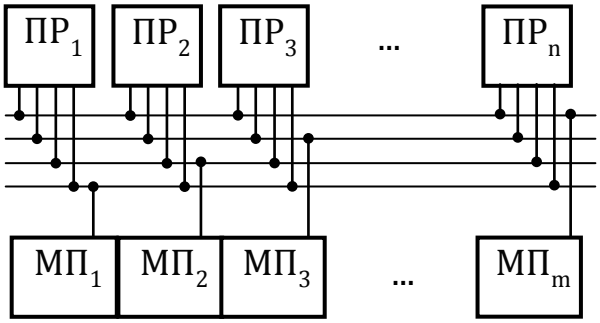


Рис. 1.15. МВС с общей памятью

1.4.3. Организация схем коммутации в МВС с распределенной памятью

На рис. 1.16 представлена типовая архитектура МВС с распределенной памятью.

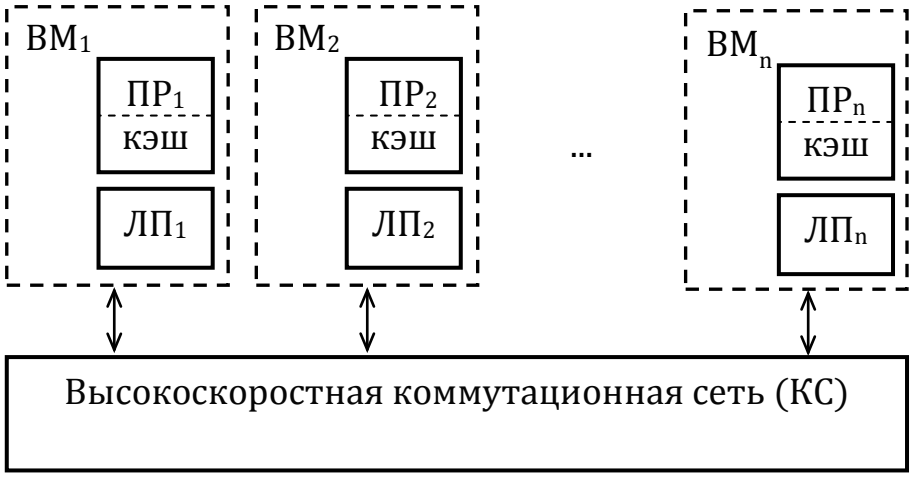


Рис. 1.16. Типовая архитектура схем коммутации в МВС с распределенной памятью

При обсуждении этого типа архитектур будем использовать следующие обозначения: число шин – *m*, число процессоров – *n*. Также будем учитывать

ограничение: $m < \frac{n}{2}$. Средства коммутации для этого типа архитектуры представляют важнейший структурный элемент ВС.

Наиболее распространенные решения при построении КС для МВС с распределенной памятью можно разбить на три группы.

1. ВС со связями через общую шину (см. рис. 1.17).

Такая схема является, с одной стороны, дешевой и просто реализуемой, а также соответствует структуре передачи данных при решении многих вычислительных задач. Легко наращивается число подключаемых вычислительных модулей (ВМ). С другой стороны, для обеспечения эффективной работы шины требуется тщательное планирование использования шины (арбитр шины), работающей в режиме разделения времени, от которой зависит производительность всей системы.

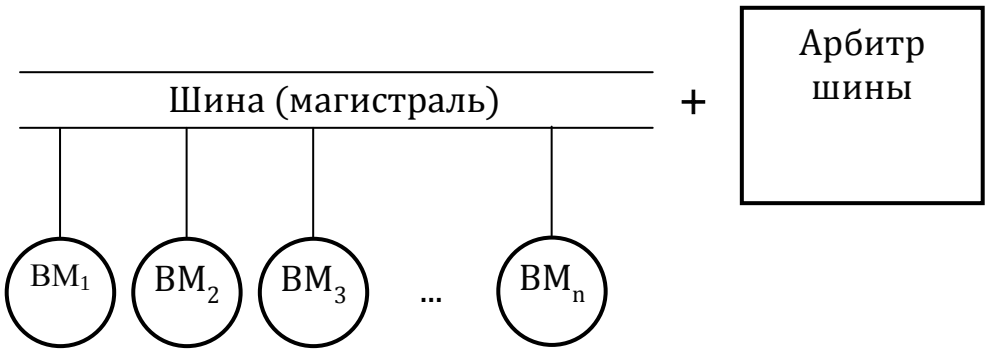


Рис. 1.17. Архитектура схем коммутации в МВС с распределенной памятью со связями через общую шину

2. Более дорогой вариант – архитектура со связями через несколько шин – представлена на рис. 1.18. При таком способе связи поддерживается режим прямого доступа к памяти, причем передача производится обычно блоками из фиксированного набора слов.

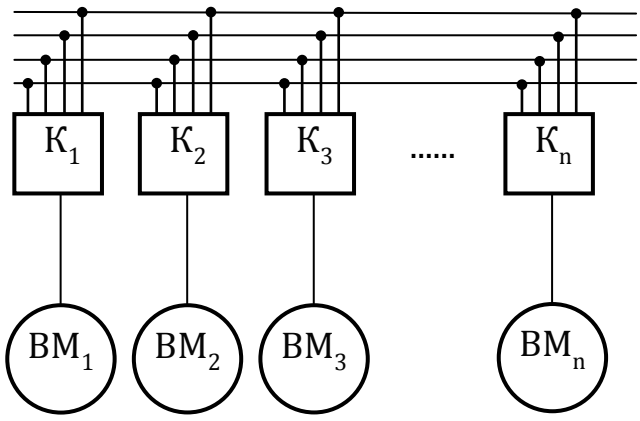


Рис. 1.18. Архитектура схем коммутации в МВС с распределенной памятью со связями через несколько шин

Достоинством архитектур МВС, использующих при коммутации несколько шин, является бо'льшая производительность и надежность, чем у аналогов с одной

шиной. Однако для организации эффективных вычислений необходимы *n* коммутаторов шин, а это высокие аппаратные затраты.

3. На рис. 1.19 представлена архитектура со связями через многоступенчатый переключатель.

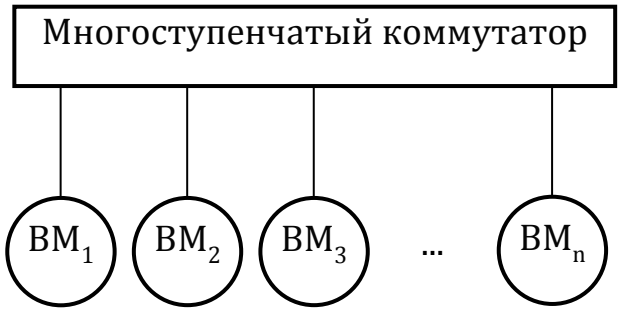


Рис. 1.19. МВС с распределенной памятью со связями через многоступенчатый переключатель

Существует большое разнообразие в организации коммутации такого вида.

3.1. При соединении первого и последнего процессоров линейки через коммутатор получается топология, называемая **кольцом** (рис. 1.20).

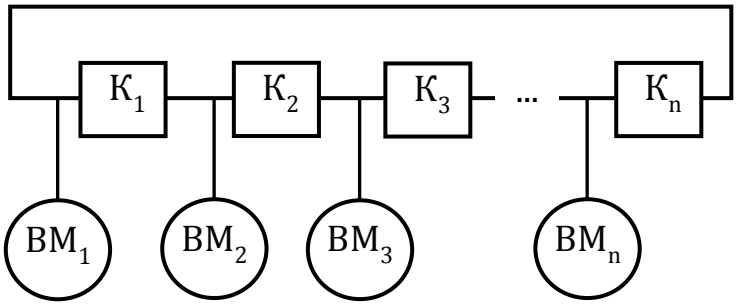


Рис. 1.20. Многоступенчатый коммутатор кольцевого типа

3.2. Система, в которой каждый ВМ связан с каждым другим ВМ прямой линией связи, построена на основе КС с **полным набором связей** (рис. 1.21). Такая топология обеспечивает высокую надежность и минимальные затраты при передаче данных, однако является сложно реализуемой при большом количестве процессов. Каждый узел – это ВМ со своим многовходовым коммутатором.

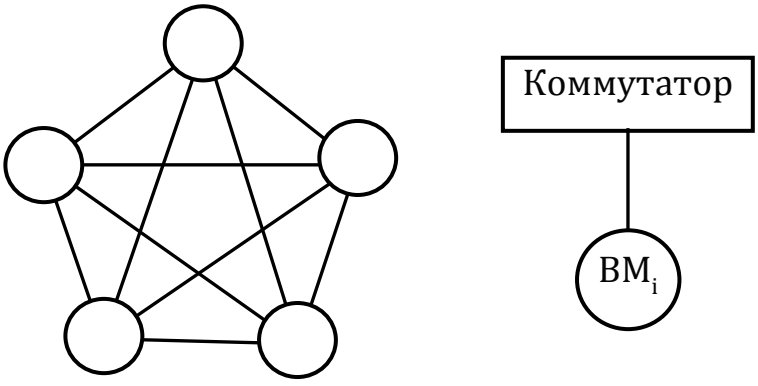


Рис. 1.21. Схема КС с полным набором связей

3.3. Система, в которой все ВМ с помощью своих коммутаторов имеют линии связи с некоторым центральным коммутатором или центральным коммутационным узлом (ЦКУ) или управляющим процессором, называется ВС с топологией типа «**звезда»**.

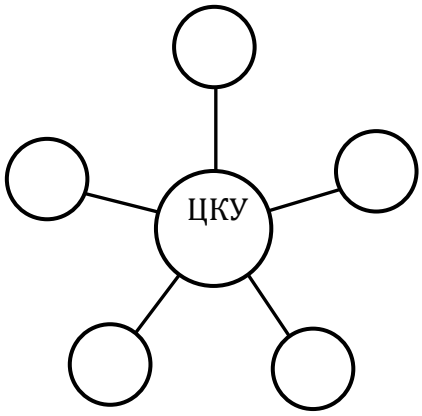


Рис. 1.22. Коммутатор по типу «звезды»

3.4. ВС, в которой топология линий связи образует прямоугольную сетку (обычно двух- или трехмерную), называется системой с **топологией решетки**. Подобная топология может быть достаточно просто реализована и, кроме того, эффективно использована при параллельном выполнении многих численных алгоритмов (например, при реализации методов анализа математических моделей, описываемых дифференциальными уравнениями в частных производных). На рис. 1.23 представлена топология связей по типу «двумерной решетки». Каждый ВМ связан с 4-мя соседями и через них с любыми другими ВМ.

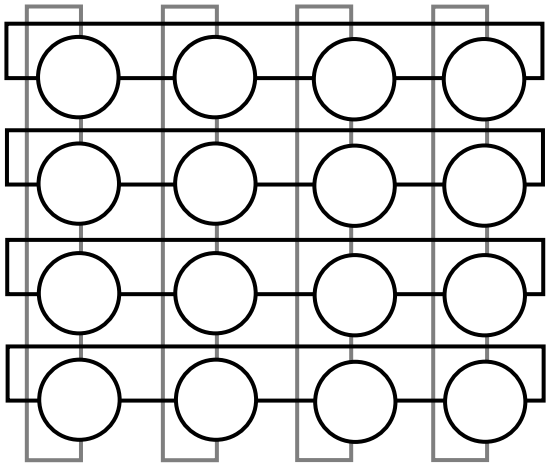


Рис. 1.23. Структура ВС по типу «решетки»

3.5. Топология ВС по типу **«гиперкуб»** – частный случай решетки, когда по каждой размерности сетки имеется только два процессора. Примеры гиперкуб-коммутаторов представлены на рис. 1.24. Гиперкуб содержит 2^N процессоров при размерности гиперкуба N.

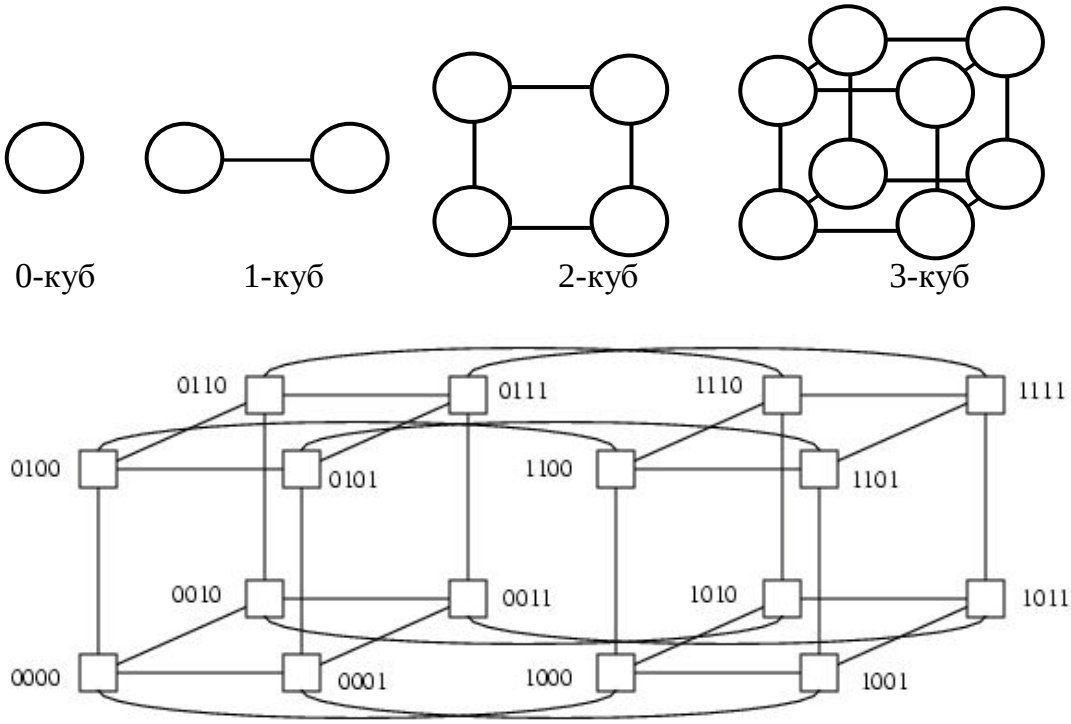


Рис. 1.24. Коммутатор по типу «гиперкуба»

Такой вариант организации сети передачи данных достаточно широко распространен на практике, достаточно прост в построении, и характеризуется следующими свойствами:

1. Два процессора имеют соединение, если двоичные представления их номеров имеют только одну различающуюся позицию.
2. В *N*-мерном гиперкубе каждый процессор связан ровно с *N* соседями.
3. Кратчайший путь между двумя любыми процессорами имеет длину, совпадающую с количество различающихся битовых значений в номерах процессов. Отсюда следует, что максимальная длина пути в *N*-мерном гиперкубе равна *N*.
4. N-мерный гиперкуб может быть разделен на два (*N*–1)-мерных гиперкуба (всего возможно *N* таких разбиений).

Очевидные преимущества такой топологии:

1. ВМ, располагаясь в вершине *N* куба, не отстоит более чем на *N*-ребер ни от какого другого ВМ, что значительно облегчает создание эффективных коммуникаций в системе (например, для *N*=12 допустимое число ВМ 2^{12} =4096!);
2. т.к. структура соединений в *N*-кубе хорошо согласуется с двоичной логикой, то достаточно легко реализуется алгоритм маршрутизации для передачи сообщений между узлами;
3. между любой парой ВМ существует несколько альтернативных путей коммуникаций, что позволяет в целом снизить задержки при передачах.

1.4.4. Архитектура систем со смешенной организацией памяти

Для решения практических задач часто используют вычислительные системы со смешанной организацией памяти. На рис. 1.25 схематично представлен пример такой архитектуры.

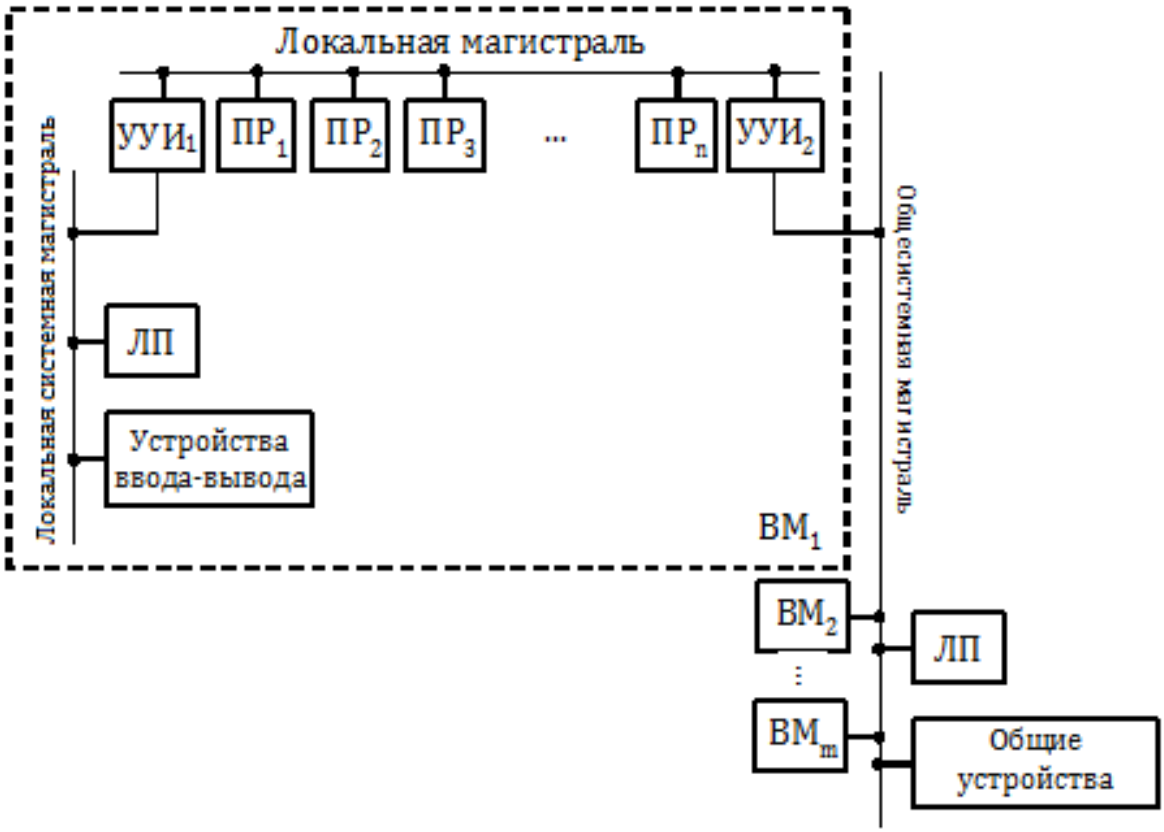


Рис. 1.25. Типовая архитектура систем со смешанной организацией памяти

Здесь:
$УУИ_1$, $УУИ_2$ – устройства управления интерфейсами;
ПР – процессор;
ЛП – локальная память;
ВМ – вычислительный модуль.

P.S. См. презентации по теме из папки «Дополнительные материалы»

Лекция 2. Архитектуры процессора

Основные вопросы:

2.1. RISC- и CISC-архитектуры, основные принципы построения и реализации

2.2. Методы адресации и типы машинных команд. Оптимизация системы команд

2.3. Компьютеры со стековой архитектурой

2.4. Микропроцессоры

2.4.1. Основные характеристики

2.4.2. Структура базового микропроцессора. Взаимодействие элементов

2.4.3. Поколения микропроцессоров семейства Intel

Архитектура ЭВМ – это **многоуровневая иерархия аппаратурно-программных средств**, из которых строится ЭВМ. Каждый из уровней допускает многовариантное построение и применение. Конкретная реализация уровней определяет особенности структурного построения ЭВМ.

В **широком смысле архитектура** охватывает понятие организации системы, включающее такие аспекты разработки компьютера как **систему памяти, структуру системной шины, организацию ввода/вывода и т.п.**

В **узком смысле** под архитектурой понимают **архитектуру набора команд**.

2.1. RISC- и CISC-архитектуры, основные принципы построения и реализации

На современном этапе развития вычислительной техники существуют две основные архитектуры набора команд, используемые компьютерной промышленностью, - архитектуры **CISC** (**C**omplete **I**nstruction **S**et **C**omputer) и **RISC** (**R**estricted (reduced) **I**nstruction **S**et **C**omputer).

Основоположником CISC-архитектуры – архитектуры с полным набором команд можно считать фирму IBM с ее базовой архитектурой IBM/360, ядро которой используется с 1964 г. и дошло до наших дней, например, в таких современных мейнфреймах, как IBM ES/9000.

Лидером в разработке микропроцессоров с полным набором команд считается компания Intel с микропроцессорами X86 и Pentium. Это практически стандарт для рынка микропроцессоров.

При проектировании суперминикомпьютеров на базе последних достижений СБИС-технологии оказалось невозможным полностью перенести в нее архитектуру удачного компьютера, выполненного на другой элементной базе. Такой перенос был бы очень неэффективен из-за технических ограничений на ресурсы кристалла: площадь, количество транзисторов, мощность рассеивания и т. д.

Для снятия указанных ограничений в Беркли (США, Калифорния) была разработана регистро-ориентированная RISC – архитектура. Компьютеры с такой архитектурой иногда называют компьютерами с сокращенным набором команд. Суть ее состоит в выделении наиболее употребительных операций и создании

архитектуры, приспособленной для их быстрой реализации. Это позволило в условиях ограниченных ресурсов разработать компьютеры с высокой пропускной способностью.

В компьютерной индустрии наблюдается настоящий бум систем с RISC-архитектурой. Рабочие станции и серверы, созданные на базе концепции RISC, завоевали лидирующие позиции благодаря своим исключительным характеристикам и уникальным свойствам операционных систем типа UNIX, используемых на этих платформах.

Четыре основных принципа RISC-архитектуры:

- каждая команда независимо от ее типа выполняется за один машинный цикл, длительность которого должна быть максимально короткой;
- все команды должны иметь одинаковую длину и использовать минимум адресных форматов, что резко упрощает логику центрального управления процессором;
- обращение к памяти происходит только при выполнении операций записи и чтения, вся обработка данных осуществляется исключительно в регистровой структуре процессора;
- система команд должна обеспечивать поддержку языка высокого уровня (имеется в виду подбор системы команд, наиболее эффективной для различных языков программирования).

Со временем трактовка некоторых из этих принципов претерпела изменения. В частности, возросшие возможности технологии позволили существенно смягчить ограничение состава команд: вместо полусотни инструкций, использовавшихся в архитектурах первого поколения, современные RISC-процессоры реализуют около 150 инструкций. Однако основной закон RISC был и остается незыблемым: обработка данных должна вестись только в рамках регистровой структуры и только в формате команд "регистр – регистр – регистр".

В RISC-микропроцессорах значительную часть площади кристалла занимает тракт обработки данных, а секции управления и дешифратору отводится очень небольшая его часть.

Аппаратная поддержка выбранных операций, безусловно, сокращает время их выполнения, однако критерием такой реализации является повышение общей производительности компьютера в целом и его стоимость. Поэтому при разработке архитектуры необходимо проанализировать результаты компромиссов между различными подходами, различными наборами операций и на их основе выбрать оптимальное решение.

Простота архитектуры RISC-процессора обеспечивает его компактность, практическое отсутствие проблем с охлаждением кристалла, чего нет в процессорах фирмы Intel, упорно придерживающейся пути развития архитектуры CISC. Формирование стратегии CISC-архитектуры произошло за счет технологической возможности перенесения "центра тяжести" обработки данных с программного уровня системы на аппаратный, так как основной путь повышения эффективности для CISC-компьютера виделся, в первую очередь, в упрощении компиляторов и минимизации исполняемого модуля. На сегодняшний день CISC-процессоры почти монопольно занимают на компьютерном рынке сектор

персональных компьютеров, однако RISC-процессорам нет равных в секторе высокопроизводительных серверов и рабочих станций.

Основные черты RISC-архитектуры в сравнении с аналогичными по характеру чертами CISC-архитектуры отображаются в табл. 2.1:

Таблица 2.1.

Основные черты архитектуры

CISC-архитектура	RISC-архитектура
Многобайтовые команды	Однобайтовые команды
Малое количество регистров	Большое количество регистров
Сложные команды	Простые команды
Одна или менее команд за один цикл процессора	Несколько команд за один цикл процессора
Традиционно одно исполнительное устройство	Несколько исполнительных устройств

Одним из важных преимуществ RISC-архитектуры является высокая скорость арифметических вычислений. RISC-процессоры первыми достигли планки наиболее распространенного стандарта IEEE 754, устанавливающего 32-разрядный формат для представления чисел с фиксированной точкой и 64-разрядный формат "полной точности" для чисел с плавающей точкой. Высокая скорость выполнения арифметических операций в сочетании с высокой точностью вычислений обеспечивает RISC-процессорам безусловное лидерство по быстродействию в сравнении с CISC-процессорами.

Другой особенностью RISC-процессоров является комплекс средств, обеспечивающих безостановочную работу арифметических устройств: механизм динамического прогнозирования ветвлений, большое количество оперативных регистров, многоуровневая встроенная кэш-память.

Организация регистровой структуры – основное достоинство и основная проблема RISC. Практически любая реализация RISC-архитектуры использует трехместные операции обработки, в которых результат и два операнда имеют самостоятельную адресацию – $R1:=R2, R3$. Это позволяет без существенных затрат времени выбрать операнды из адресуемых оперативных регистров и записать в регистр результат операции. Кроме того, трехместные операции дают компилятору большую гибкость по сравнению с типовыми двухместными операциями формата "регистр – память" архитектуры CISC. В сочетании с быстродействующей арифметикой RISC-операции типа "регистр – регистр" становятся очень мощным средством повышения производительности процессора.

Вместе с тем опора на регистры является ахиллесовой пятой RISC-архитектуры. Проблема в том, что в процессе выполнения задачи RISC-система неоднократно вынуждена обновлять содержимое регистров процессора, причем за минимальное время, чтобы не вызывать длительных простоев арифметического устройства. Для CISC-систем подобной проблемы не существует, поскольку модификация регистров может происходить на фоне обработки команд формата "память – память".

Существуют два подхода к решению проблемы модификации регистров в RISC-архитектуре: аппаратный, предложенный в проектах RISC-1 и RISC-2, и программный, разработанный специалистами IBM и Стэндфордского университета. Принципиальная разница между ними заключается в том, что аппаратное решение основано на стремлении уменьшить время вызова процедур за счет установки дополнительного оборудования процессора, тогда как программное решение базируется на возможностях компилятора и является более экономичным с точки зрения аппаратуры процессора.

2.2. Методы адресации и типы машинных команд. Оптимизация системы команд

В машинах с регистрами общего назначения метод (или режим) адресации объектов, с которыми манипулирует команда, может задавать константу, регистр или ячейку памяти.

В табл. 2.2 представлены основные методы адресации операндов.

Таблица 2.2.

Методы адресации

Метод адресации	**Пример команды**	**Смысл команды**	**Использование команды**
Регистровая	*Add R*4, *R*3	*R*4 = *R*4+*R*3	Для записи требуемого значения в регистр
Непосредственная или литерная	*Add R*4, #3	*R*4 = *R*4+3	Для задания констант
Базовая со смещением	*Add R*4, 100*(R*1*)*	*R*4= *R*4+*M(*100+*R*1*)*	Для обращения к локальным переменным
Косвенная регистровая	*Add R*4, *(R*1*)*	*R*4 = *R*4+*M(R*1*)*	Для обращения по указателю к вычисленному адресу
Индексная	*Add R*3, *(R*1+*R*2*)*	*R*3 = *R*3+*M(R*1+*R*2*)*	Полезна при работе с массивами: R1 – база, R3 – индекс
Прямая или абсолютная	*Add R*1, *(*1000*)*	*R*1=*R*1+*M(*1000*)*	Полезна для обращения к статическим данным
Косвенная	*Add R*1, *@(R*3*)*	*R*1 = *R*1+*M(R*3*)*	Если *R*3 – адрес указателя *p*, то выбирается значение по этому указателю

Автоинкрементная	*Add R*1, *(R*2*)+*	*R*1 = *R*1+*M(R*2*)* *R*2 = *R*2+*d*	Полезна для прохода в цикле по массиву с шагом: *R*2 – начало массива. В каждом цикле *R*2 получает приращение *d*
Автодекрементная	*Add R*1, *(R*2*)–*	*R*2 = *R*2–*d* *R*1 = *R*1+*M(R*2*)*	Аналогична предыдущей; обе могут использоваться для реализации стека
Базовая индексная со смещением и масштабированием	*Add* *R*1,100*(R*2*)(R*3*)*	*R*1=*R*1+*M(*100*)*+*R*2+ *R*3**d*	Для индексации массивов

Адресация непосредственных данных и литерных 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).

В табл. 2.2 на примере команды сложения (*Add*) приведены наиболее употребительные названия методов адресации, хотя при описании архитектуры в документации производители компьютеров и программного обеспечения используют разные названия для этих методов. В табл. 2.2 знак "=" используется для обозначения оператора присваивания, а буква *M* обозначает память (*Memory*). Таким образом, *M(R*1*)* обозначает содержимое ячейки памяти, адрес которой определяется содержимым регистра *R*1.

Использование сложных методов адресации позволяет существенно сократить количество команд в программе, но при этом значительно увеличивается сложность аппаратуры.

Команды традиционного машинного уровня можно разделить на несколько типов, которые показаны в табл. 2.3.

Таблица 2.3.

Основные типы команд

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 из процедур
Системные операции	Системные вызовы, команды управления виртуальной памятью и т. д.
Операции с плавающей	Операции сложения, вычитания, умножения и деления

точкой	над вещественными числами
Десятичные операции	Десятичное сложение, умножение, преобразование форматов и т. д.
Операции над строками	Пересылки, сравнения и поиск строк

Тип операнда может задаваться либо кодом операции в команде, либо с помощью тега, который хранится вместе с данными и интерпретируется аппаратурой во время обработки данных.

Обычно тип операнда (целый, вещественный, символ) определяет и его размер. Как правило, целые числа представляются в дополнительном коде. Для задания символов компания IBM использует код EBCDIC, другие компании применяют код ASCII. Для представления вещественных чисел с одинарной и двойной точностью придерживаются стандарта IEEE 754.

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

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 программы с лихвой окупается возможностью простого увеличения частоты RISC-процессоров. Этот процесс можно наблюдать сегодня, когда максимальные тактовые частоты практически всех RISC-процессоров (Alpha, R4400, HyperSPARC и Power2) превышают тактовую частоту, достигнутую процессором Pentium.

Общую технологию проектирования системы команд (СК) для новой ЭВМ можно обозначить так: зная класс решаемых задач, выбираем некоторую типовую СК для широко распространенного компьютера и исследуем ее на предмет присутствия всего разнообразия операций в заданном классе задач. Вовсе не встречающиеся или редко встречающиеся операции не покрываем командами. Все частоты встреч операций для задания их в СК всякий раз можно определить из соотношений "стоимость затрат – сложность реализации – получаемый выигрыш".

Второй путь проектирования СК состоит в расширении имеющейся системы команд. Один из способов такого расширения – создание макрокоманд, второй – используя имеющийся синтаксис языка СК, дополнить его новыми командами с последующим переассемблированием, через расширение функций ассемблера. Оба эти способа принципиально одинаковы, но отличаются в тактике реализации аппарата расширения.

Так, система команд для ПК IBM покрывает следующие группы операций:

- передачи данных,
- арифметические операции,
- операции ветвления и циклов,
- логические операции
- операции обработки строк.

Разработанную СК следует **оптимизировать**. Один из способов **оптимизации состоит в выявлении частоты повторений сочетаний двух или более команд,** следующих друг за другом в некоторых типовых задачах для данного компьютера, **с последующей заменой их одной командой, выполняющей те же функции.** Это приводит к сокращению времени выполнения программы и уменьшению требуемого объема памяти.

Можно также исследовать и часто генерируемые компилятором некоторые последовательности команд, убирая из них избыточные коды.

Оптимизацию можно проводить и в пределах отдельной команды, исследуя ее информационную емкость. Для этого можно применить аппарат теории информации, в частности для оценки количества переданной информации – энтропию источника. Тракт "процессор – память" можно считать каналом связи.

Замечание. **Энтропия** – это мера вероятности пребывания системы в данном состоянии (в статистической физике).

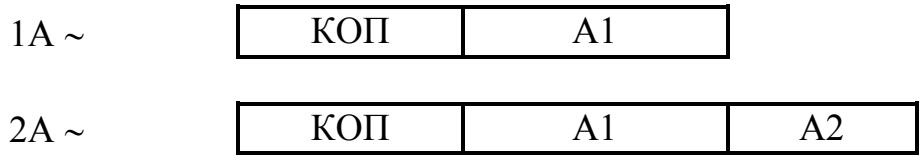
2.3. Компьютеры со стековой архитектурой

При создании компьютера одновременно проектируют и систему команд для него. Существенное влияние на выбор операций для их включения в СК оказывают:

- **элементная база и технологический уровень производства компьютеров;**
- **класс решаемых задач**, определяющий необходимый набор операций, воплощаемых в отдельные команды;
- **системы команд** для компьютеров аналогичного класса;
- **требования к быстродействию** обработки данных, что может породить создание команд с большой длиной слова (VLIW-команды).

Анализ задач показывает, что в смесях программ доминирующую роль играют команды пересылки и процессорные команды, использующие регистры и простые режимы адресации.

На сегодняшний день наибольшее распространение получили следующие структуры команд: одноадресные (1А), двухадресные (2А), трехадресные (3А), безадресные (БА), команды с большой длиной слова (VLIW – БДС) (рис. 2.1):



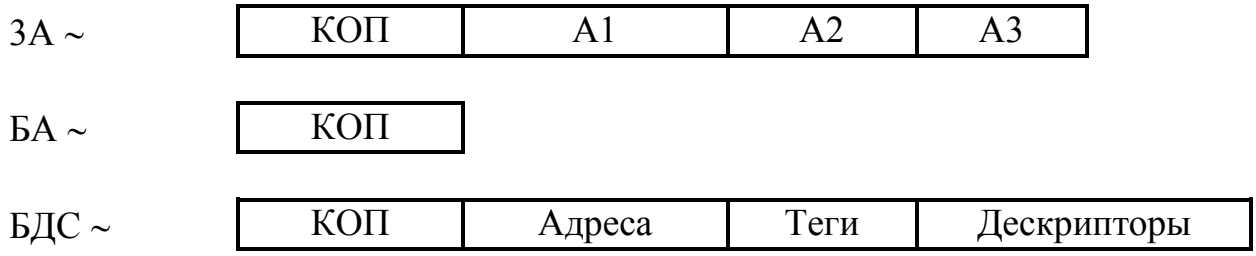


Рис. 2.1. Структуры команд

Причем операнд может указываться как адресом, так и непосредственно в структуре команды. В случае БА-команд операнды выбираются из стека, результаты помещаются в стек (магазин, гнездо). Типичными первыми представителями БА-компьютеров являются KDF-9 и МВК "Эльбрус". Их характерной особенностью является наличие стековой памяти.

Стек – это область памяти, которая используется для временного хранения данных и операций. Доступ к элементам стека осуществляется по принципу FILO (first in, last out) – первым вошел, последним вышел. Кроме того, доступ к элементам стека осуществляется только через его вершину, т. е. пользователю "виден" лишь тот элемент, который помещен в стек последним.

Рассмотрим функционирование процессора со стековой организацией памяти.

Вначале рассмотрим пример вычисления значения выражения

$$X = \frac{a^2 + b^2}{b + c}$$

с помощью одноадресного компьютера. Последовательность необходимых команд приведена в табл. 2.4.

Таблица 2.4.

Последовательность команд для вычисления выражения

Номер команды	**Команда**	**Комментарии**
1	$c \to \sum$	*Выборка из памяти переменной* ***c*** *и загрузка в регистр* $\boldsymbol{\Sigma}$
2	$(\sum) + b$	*Выборка из памяти переменной* ***b****, выполнение операции сложения с содержимом регистра* $\boldsymbol{\Sigma}$ *и сохранение результата в регистре* $\boldsymbol{\Sigma}$
3	$(\sum) \to P_1$	*Сохранение результата сложения* в P_1 *– рабочей ячейке в памяти*
4	$a \to \sum$	*Выборка из памяти переменной* ***a*** *и загрузка в регистр* $\boldsymbol{\Sigma}$
5	$(\sum) \bullet a$	*Выполнение операции умножения с сохранением результата в регистре*
6	$(\sum) \to P_2$	*Сохранение результата сложения в* P_2 *– рабочей ячейке*
7	$b \to \sum$	*Выборка из памяти переменной* ***b*** *и загрузка в регистр* $\boldsymbol{\Sigma}$

8	$(\Sigma)\bullet b$	*Выполнение операции умножения с сохранением результата в регистре*
9	$(\Sigma)+(P_2)$	*и т.д.*
10	$(\Sigma):(P_1)$	$X=\dfrac{a^2+b^2}{b+c}\to\Sigma$

Замечание. Выполнение команды типа $(\Sigma\ )\otimes(\mathbf{P_i})$ подразумевает, что результат операции помещается в первый регистр, в данном случае в регистр $\boldsymbol{\Sigma}$

Если главным фактором, ограничивающим быстродействие компьютера, является время цикла памяти, то необходимость в дополнительных обращениях к памяти значительно снижает скорость его работы. Очевидно, что принципиально необходимы только обращения к памяти за данными в первый раз. В дальнейшем они могут храниться в триггерных регистрах или кеш памяти.

Указанные соображения получили свое воплощение в ряде логических структур процессора. Одна из них – процессор со стековой памятью. Принцип ее работы поясняет схема, представленная на рис. 2.2.

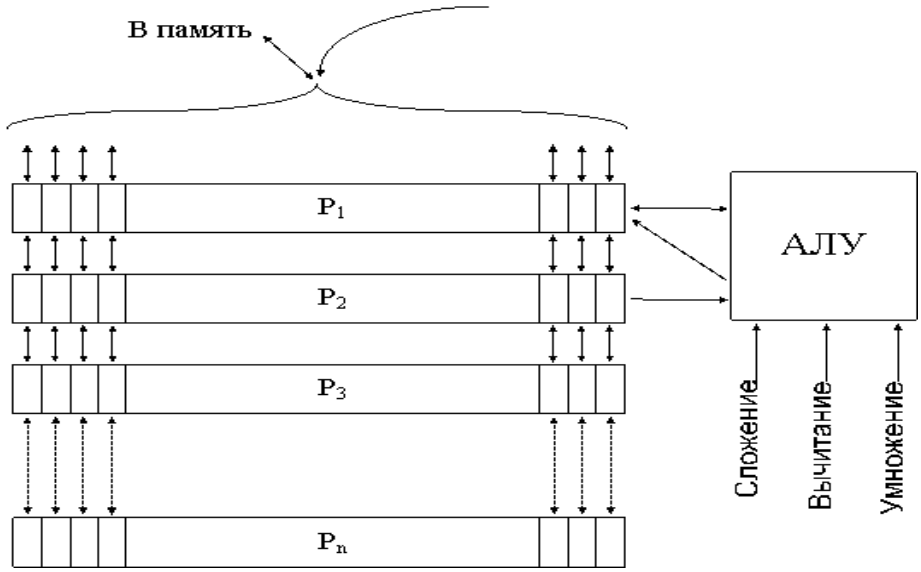


Рис. 2.2. Стековая организация процессора

Стековая память представляет собой набор из $\boldsymbol{n}$ регистров, каждый из которых способен хранить одно машинное слово. Одноименные разряды регистров $\boldsymbol{P_1, P_2, ..., P_n}$ соединены между собой цепями сдвига. Поэтому весь набор регистров может рассматриваться как группа $\boldsymbol{n}$-разрядных сдвигающих регистров, составленных из одноименных разрядов регистров $\boldsymbol{P_1, P_2, ..., P_n}$. Информация в стеке может продвигаться между регистрами вверх и вниз.

Движение вниз: $(P_1)\to P_2$, $(P_2)\to P_3$, ..., а $\boldsymbol{P_1}$ заполняется данными из главной памяти. Движение вверх: $(P_n)\to P_{n-1}$, $(P_{n-1})\to P_{n-2}$, а $\boldsymbol{P_n}$ заполняется нулями.

Регистры $\boldsymbol{P_1}$ и $\mathbf{P_2}$ связаны с АЛУ, образуя два операнда для выполнения операции. Результат операции записывается в $\boldsymbol{P_1}$. Следовательно, АЛУ выполняет

операцию $(P_1)\otimes(P_2)\to P$. Одновременно с выполнением арифметической операции осуществляется продвижение операндов вверх, не затрагивая **P_1**, т.е. $(P_3)\to P_2$, $(P_4)\to P_3$ и т.д.

Таким образом, арифметические операции используют подразумеваемые адреса, что уменьшает длину команды. В принципе, в команде достаточно иметь только поле, определяющее код операции. Поэтому компьютеры со стековой памятью называют безадресными. В то же время команды, осуществляющие вызов или запоминание информации из главной памяти, требуют указания адреса операнда. Поэтому в компьютерах со стековой памятью используются команды переменной длины.

Команды располагаются в памяти в виде непрерывного массива слогов независимо от границ ячеек памяти. Это позволяет за один цикл обращения к памяти вызвать несколько команд.

Для эффективного использования возможностей такой памяти вводятся специальные команды:

•***дублирование*** ~ **$(P_1)\to P_2$, $(P_2)\to P_3$,** ... и т. д., а (**P_1**) остается при этом неизменным;

•***реверсирование*** ~ **$(P_1)\to P_2$, а $(P_2)\to P_1$**, что удобно для выполнения некоторых операций.

Рассмотрим предыдущий пример вычисления выражения, но с использованием процессора со стековой организацией памяти (табл. 2.5):

$$X=\frac{a^2+b^2}{b+c}$$

Таблица 2.5.

Последовательность команд для процессора со стековой памятью

№	Команда	P_1	P_2	P_3	P_4
1	*Вызов b*	*b*			
2	*Дублирование*	*b*	*b*		
3	*Вызов c*	*c*	*b*	*b*	
4	*Сложение*	*b+c*	*b*		
5	*Реверсирование*	*b*	*b+c*		
6	*Дублирование*	*b*	*b*	*b+c*	
7	*Умножение*	b^2	*b+c*		
8	*Вызов a*	*a*	b^2	*b+c*	
9	*Дублирование*	*a*	*a*	b^2	*b+c*
10	*Умножение*	a^2	b^2	*b+c*	
11	*Сложение*	a^2+b^2	*b+c*		
12	*Деление*	$\frac{a^2+b^2}{b+c}$			

Как следует из табл. 2.5, понадобились лишь **три обращения к памяти** для вызова операндов (команды 1, 3, 8). Меньше обращений принципиально невозможно. Операнды и промежуточные результаты поступают для операций в АЛУ из стековой памяти; 9 команд из 12 являются безадресными.

Вся программа размещается в **трех 48-разрядных ячейках памяти.**

Главное преимущество использования магазинной памяти состоит в том, что при переходе к подпрограммам (ПП) или в случае прерывания нет необходимости в специальных действиях по сохранению содержимого арифметических регистров в памяти. Новая программа может немедленно начать работу. При введении в стековую память новой информации данные, соответствующие предыдущей программе, автоматически продвигаются вниз и возвращаются обратно, когда новая программа закончит вычисления.

Наряду с указанными **преимуществами** стековой памяти отметим также:

- уменьшение количества обращений к памяти;
- упрощение способа обращения к ПП и обработки прерываний.

Недостатки стековой организации памяти:

- большое число регистров с быстрым доступом;
- необходимость в дополнительном оборудовании, чтобы следить за переполнением стековой памяти, ибо число регистров памяти конечно;
- приспособленность главным образом для решения научных задач и в меньшей степени для систем обработки данных или управления технологическими процессами.

2.4. Микропроцессоры

Основу центрального процессора современной ЭВМ составляет микропроцессор – обрабатывающее устройство, служащее для арифметических и логических преобразований данных, для организации обращения к ОП и ВнУ и для управления ходом вычислительного процесса.

В настоящее время существует большое число разновидностей микропроцессоров, различающихся назначением, функциональными возможностями, структурой, исполнением. Чаще всего наиболее существенным, классификационным различием между ними является количество разрядов в обрабатываемой информационной единице: 8-битовые, 16-битовые, 32-битовые, 64-битовые и появившиеся недавно 128-битовые.

К группе 8-битовых микропроцессоров относятся i8080, i8085 (с буквы i начинаются названия МП, выпускаемых фирмой Intel - INTegrated Electronics), Z80 (с буквы Z начинаются названия МП фирмы Zilog) и др.

Наибольшее распространение среди 16-битовых микропроцессоров получили i8086, i8088, 32-битовых - i80386, i80486, которые совместимы по видам и форматам данных снизу вверх. Эти микропроцессоры используются в различных модификациях IBM PC.

2.4.1. Основные характеристики

Тактовая частота - это количество электрических импульсов в секунду, определяет максимальное время выполнения переключения элементов в ЭВМ;

Разрядность - максимальное число одновременно обрабатываемых двоичных разрядов.

Разрядность МП обозначается **m/n/k** и включает:

m - разрядность внутренних регистров, определяет принадлежность к тому или иному классу процессоров;

n - разрядность шины данных, определяет скорость передачи информации;

k - разрядность шины адреса, определяет размер адресного пространства.

Например, МП i8088 характеризуется значениями m/n/k=16/8/20;

Современные микропроцессоры построены на 32-х битной архитектуре x86 или IA-32 (Intel Architecture 32 bit), или на более совершенной и производительной 64-х битную архитектуру IA-64 (Intel Architecture 64 bit). В 2003 г. - массовый выпуск и выход в продажу нового микропроцессора Athlon 64 корпорации AMD (Advanced Micro Devices), который примечателен тем, что может работать как с 32-х битными приложениями, так и с 64-х битными. Производительность 64-х битных микропроцессоров намного выше.

Архитектура

Понятие архитектуры микропроцессора включает в себя систему команд и способы адресации, возможность совмещения выполнения команд во времени, наличие дополнительных устройств в составе микропроцессора, принципы и режимы его работы. Выделяют понятия **микро** и **макро архитектуры.**

Микро архитектура микропроцессора - это аппаратная организация и логическая структура микропроцессора, регистры, управляющие схемы, арифметико-логические устройства, запоминающие устройства и связывающие их информационные магистрали.

Макро архитектура - это система команд, типы обрабатываемых данных, режимы адресации и принципы работы микропроцессора.

В общем случае под архитектурой ЭВМ понимается абстрактное представление машины в терминах основных функциональных модулей, языка ЭВМ, структуры данных.

В соответствии с архитектурными особенностями, определяющими свойства системы команд, различают:

Микропроцессоры с CISC архитектурой.

CISC (Complex Instruction Set Computer) - Компьютер со сложной системой команд. Исторически они первые и включают большое количество команд. Все микропроцессоры корпораций Intel (Integrated Electronics) и AMD (Advanced Micro Devices) относятся к категории CISC.

Микропроцессоры с RISC архитектурой.

RISC (Reduced Instruction Set Computer) - Компьютер с сокращенной системой команд. Упрощена система команд и сокращена до такой степени, что каждая

инструкция выполняется за единственный такт. Вследствие этого упростилась структура микропроцессора, и увеличилось его быстродействие.

Пример микропроцессора с RISC-архитектурой - Power PC. Микропроцессор Power PC начал разрабатываться в 1981 году тремя фирмами: IBM, Motorola, Apple.

Микропроцессоры с MISC архитектурой.

MISC (Minimum Instruction Set Computer) - Компьютер с минимальной системой команд. Последовательность простых инструкций объединяется в пакет, таким образом, программа преобразуется в небольшое количество длинных команд.

Объем адресуемой памяти – максимальный объем памяти, который может обслужить микропроцессор.

32-х разрядный микропроцессор может обслужить 64 Гб ($4x10^9$ байт) памяти, а 64-х разрядный микропроцессор может обслужить 64 Тб ($64x10^{12}$ байт) памяти.

Набор дополнительных инструкций (Instruction Set) - применяется в современных CISC-микропроцессорах для ускорения работы.

Используется только при условии поддержки данных наборов со стороны приложения.
Все традиционные современные процессоры поддерживают набор инструкций MMX, который был самым первым (разработан корпорацией Intel еще в 1997 году). MMX - MultiMedia eXtensions (мультимедийные расширения, ориентированные на обработку цифрового изображения и звука. В основе технологии лежит концепция (микро архитектура) SIMD (Single Instruction Many Data – "одна команда, много данных"), когда при помощи одной инструкции одновременно обрабатывается несколько элементов данных. SSE, SSE2, 3DNow! - дальнейшее развитие этой идеи. Микропроцессоры Intel Pentium 3 поддерживают SSE, а Pentium 4 и AMD Athlon 64 еще и SSE2 (это относится и к соответствующим микропроцессорам Intel Celeron). Процессоры AMD Athlon и Duron поддерживают наборы инструкций 3DNow!Professional и MMX, в Athlon XP была добавлена поддержка SSE (на уровне микрокода ядра).

Технологический процесс производства (Process Technology) – определяет размеры элементов и соединений между ними в интегральной схеме. Измеряется в микрометрах (0,35 μm; 0,25 μm;…). Чем меньше число, тем меньше сам кристалл, следовательно, меньше потребляемая мощность и тепловыделение. Тепловыделение сильно препятствует увеличению частоты, на которой работает микропроцессор. Где-то в 1997 году произошел переход с 0,25 μm на 0,18 μm технологию производства. А уже в 2001 году произошел переход на 0,13 μm технологию, что позволило намного увеличить частоту. Сегодня уже используется технология 0,08 μm.

Производительность микропроцессора - определяется следующими параметрами:

1. ***Тактовая частота (Частота ядра)*** (Internal clock) – количество электрических импульсов в секунду. Каждый импульс несет в себе некую

информацию - это могут быть команды процессору или данные памяти. Тактовая частота задается кварцевым генератором - одним из блоков, расположенных на материнской плате. Тактовая частота кварцевого генератора выдерживается с очень высокой точностью и лежит в мега или гигагерцовом диапазоне.

Один герц - один импульс, один мегагерц - 10^6 импульсов, один гигагерц - 10^3 мегагерц. Микропроцессор, работающий на тактовой частоте 800 МГц, выполняет 800 миллионов рабочих тактов в секунду. В зависимости от сложности обрабатываемой команды процессору для выполнения задачи необходимы сотни и тысячи тактов. Но для выполнения простых операций бывает достаточно одного такта. Чем выше тактовая частота ядра, тем выше скорость обработки данных. Современные микропроцессоры работают на частотах от 300 МГц до *2,5 ГГц.*

2. ***Частота системной шины*** (System clock или Front Side Bus) – системная шина служит для связи микропроцессора с остальными устройствами. Микропроцессор имеет две частоты: тактовая частота ядра и частота системной шины. Чем выше частота системной шины, тем выше скорость передачи данных между микропроцессором и остальными устройствами. Частота системной шины современных микропроцессоров от 66 МГц до 266МГц.

3. ***Объем Кэш-памяти*** (Cache) – кэш-память быстрая память малой емкости, используемая процессором для ускорения операций, требующих обращения к памяти. Кэш – промежуточное звено между микропроцессором и оперативной памятью. Различают несколько уровней кэша: кэш первого уровня (L1) - кэш команд (инструкций), которые предстоит исполнить, кэш первого уровня размещается на одном кристалле с процессором. Кэш второго уровня (L2) - кэш данных - используется для ускорения операций с данными (в первую очередь чтения). На общую производительность влияет размер кэша L2. Чем больше L2, тем дороже процессор, т.к. память для кэша еще очень дорога. Поэтому эффективнее увеличивать частоту кэша, а для этого он должен находиться как можно ближе к ядру процессора. Кэш-память может работать на частоте 1/4, 1/3, 1/2, 1/1 от частоты ядра. Современные микропроцессоры имеют кэш объемом от 8 Кб до 5Мб. (***см. более подробно в лекции 5***)

Предельно эксплуатационные параметры микропроцессоров:

- ***Напряжение питания микропроцессора*** – величина питающего напряжения микропроцессоров зависит от технологического процесса и от частоты ядра. Чем меньше кристалл и ниже частота, тем меньше напряжение питания. Напряжение питания современных микропроцессоров от 0,5 В до 3,5 В, чаще всего от 1,2 В до 1,75 В.
- ***Ток ядра*** – у современных микропроцессоров ток, протекающий через ядро от 1 А до 90 А.
- ***Потребляемая мощность*** – зависит от величины питающего напряжения и от частоты ядра. Чем меньше напряжение питания и частота, тем меньше потребляемая мощность. Мощность современных микропроцессоров от 1Вт до 120 Вт. Чаще всего в пределах 40-70 Вт.
- ***Максимальная температура нагрева кристалла*** – максимальная температура кристалла, при которой возможна стабильная работа

микропроцессора. У современных микропроцессоров она колеблется в пределах от 60˚С до 95˚С.

Физические параметры микропроцессоров (Форм-фактор):

1. ***Тип, размеры корпуса***
2. ***Размеры кристалла***
3. ***Количество выводов***
4. ***Форма расположения выводов***

2.4.2. Структура базового микропроцессора. Взаимодействие элементов

На рис. 2.3 приведена схема базовой модели микропроцессора фирмы Intel. Условно микропроцессор можно разделить на две части: исполнительный блок (Execution Unit - EU) и устройство сопряжения с системной магистралью (Bus Interface Unit - BIU).

В исполнительном блоке находятся: арифметический блок и регистры общего назначения (РОН). Арифметический блок включает арифметико-логическое устройство, вспомогательные регистры для хранения операндов и регистр флагов.

Восемь регистров исполнительного блока МП (AX, BX, CX, DX, SP, BP, SI, DI), имеющих длину, равную машинному слову, делятся на две группы.

Первую группу составляют регистры общего назначения: AX, BX, CX и DX, каждый из которых представляет собой регистровую пару, составленную из двух регистров длиной в 0.5 машинного слова: аккумулятор, *или* регистр AX состоит из регистров AH и AL. Регистр базы (Base Register) BX состоит из регистров BH и BL. Счетчик (Count Register) CX включает регистры CH и CL. Регистр данных (Data Register) DX содержит регистры DH и DL. Каждый из коротких регистров может использоваться самостоятельно или в составе регистровой пары. Условные названия (аккумулятор, регистр базы, счетчик, регистр данных) не ограничивают применения этих регистров - эти названия говорят о наиболее частом использовании их или об особенности использования того или иного регистра в той иди иной команде.

Вторую группу составляют адресные регистры SP, BP, SI и DI (в старших моделях количество адресных регистров увеличено). Эти регистры активно используются по функциональному назначению и в других целях их применять не рекомендуется. В качестве адресного регистра часто используется РОН BX. Программно допускается использование регистров BP, DI и SI в качестве регистров для хранения операндов, но отдельные байты в этих регистрах недоступны. Основное их назначение - хранить числовые значения, реализуемые при формировании адресов операндов.

Устройство сопряжения с системной магистралью содержит управляющие регистры, конвейер команд, АЛУ команд, устройство управления исполнительным блоком МП и интерфейс памяти (соединяющий внутреннюю магистраль МП с системной магистралью ПЭВМ).

Управляющие регистры BIU: CS (указатель командного сегмента), DS указатель сегмента данных), SS (указатель сегмента стека), ES (указатель дополнительного сегмента) и др. служат для определения физических адресов ОП - операндов и команд. Регистр IP (Instruction Pointer) является указателем адреса команды, которая будет выбираться в конвейер команд в качестве очередной команды (в отечественной литературе такое устройство называется *счетчик команд).* Конвейер команд МП хранит несколько команд, что позволяет при выполнении линейных программ совместить подготовку очередной команды с выполнением текущей.

К управляющим регистрам МП относится и регистр флагов, каждый разряд которого имеет строго определенное назначение. Обычно разряды регистра флагов устанавливаются аппаратно при выполнении очередной операции в зависимости от получаемого в АЛУ результата. При этом фиксируются такие свойства получаемого результата, как нулевой результат, отрицательное число, переполнение разрядной сетки АЛУ и т.д. Но некоторые разряды регистра флагов могут устанавливаться по специальным командам. Некоторые разряды имеют чисто служебное назначение (например, хранят разряд, выпавший из АЛУ во время сдвига) или являются резервными (т.е. не используются).

Все флаги младшего байта регистра устанавливаются арифметическими или логическими операциями МП. Все флаги старших байтов, за исключением флага переполнения, устанавливаются программным путем, для этого в МП имеются команды установки флагов (STC, STD, STI), сброса (CLC CLD, CLI), инвертирования (CMC).

Работой МП управляет программа, записанная в ОП ЭВМ. Адрес очередной команды хранится в счетчике команд IP (Instruction Pointer) и в одном из сегментных регистров, чаще всего в CS. Каждый из них в реальном режиме имеет длину 16 бит, тогда как физический адрес ОП должен иметь длину 20 бит. Несогласованность длины машинного слова (16 бит) и длины физического адреса ОП (20 бит) приводит к тому, что в командах невозможно указать физический адрес ОП - его приходится формировать, собирать из разных регистров МП в процессе работы.

В реальном режиме вся ОП делится на сегменты (длина сегмента - 64 Кб). Адрес ОП разделяется на две части: номер сегмента в ОП (база сегмента) и номер ячейки внутри данного сегмента (смещение относительно начала сегмента). Базовый адрес сегмента образуется добавлением к номеру сегмента справа четырех нулей. Поскольку последние четыре разряда абсолютного (физического) адреса сегмента всегда нулевые, сегмент может начинаться не с любой ячейки ОП, а только с “параграфа” - начала 16-байтного блока ОП.

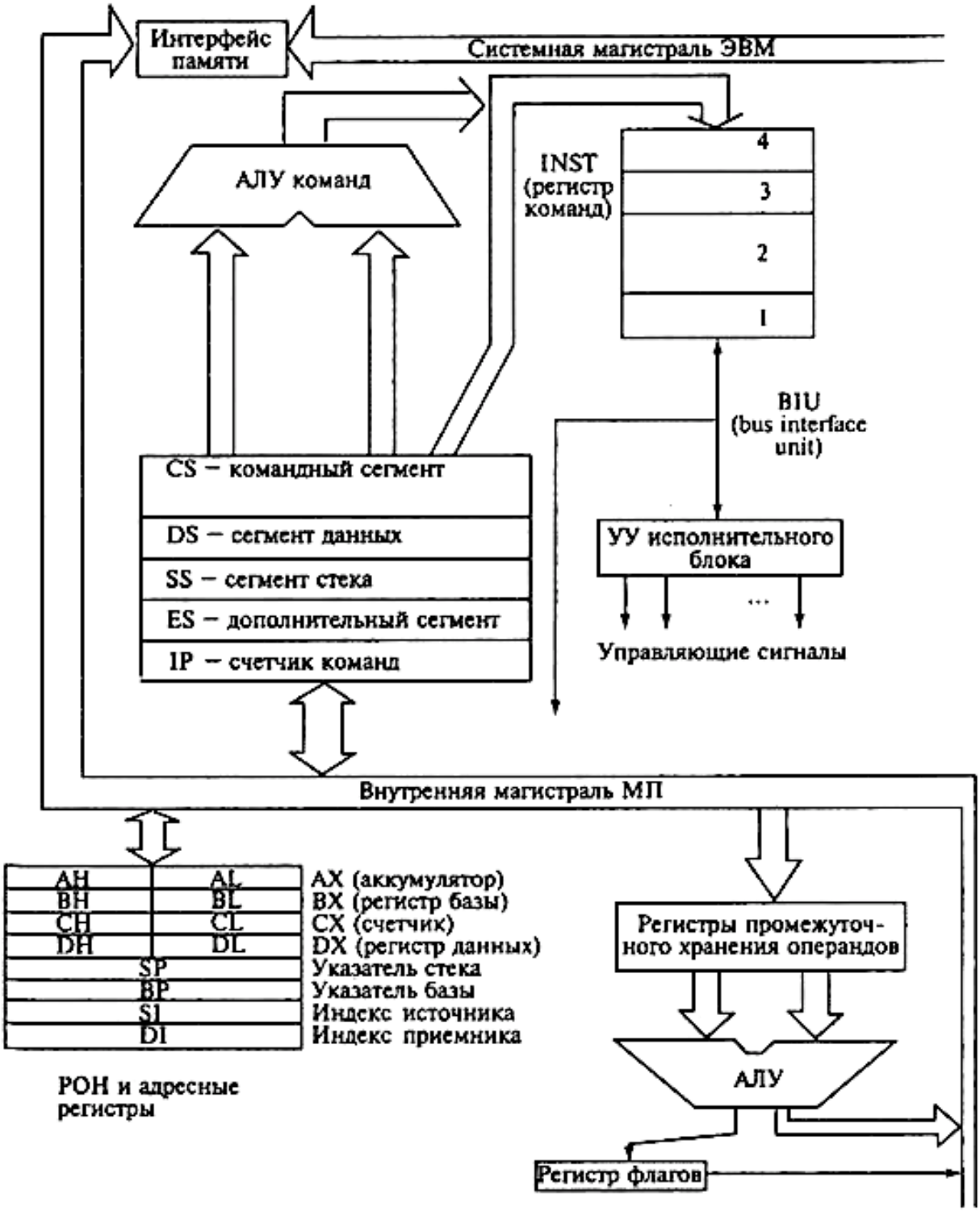


Рис. 2.3. Базовая модель микропроцессора Intel

В структуре микропроцессора имеется несколько регистров сегментов, например, в i8086 - четыре:

- C S - программный сегмент;
- DS - сегмент данных (информационный сегмент);
- SS - стековый сегмент;
- ES - расширенный сегмент (дополнительный сегмент данных).

Номер ячейки внутри сегмента (смещение) называется также *исполнительным* адресом. В большинстве случаев в адресной части команды указывается именно исполнительный адрес - номер сегмента чаще всего подразумевается по умолчанию. Однако допускается указание и полного адреса ОП в виде префиксной структуры: “сегмент: смещение”. Если сегмент в команде не указывается, значит, работа ведется внутри текущего сегмента (характер выполняемой работы и какой из сегментных регистров определяет текущую базу сегмента, зависят от вида выполняемой команды).

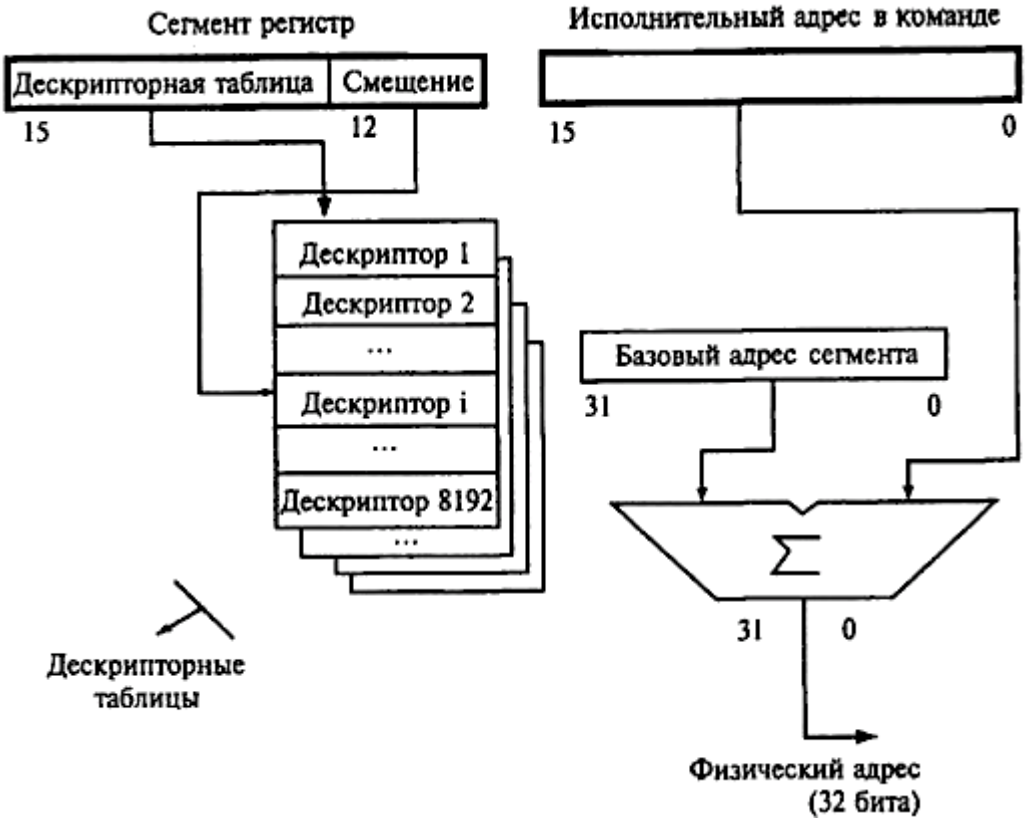


Рис. 2.4. Формирование физического адреса ОП в защищенном режиме

Номер сегмента так же, как и смещение, имеет длину 2 байта. При вычислении физического адреса ОП сегмент и смещение суммируются, но сегмент перед суммированием сдвигается влево на 4 бита. В результате суммирования образуется физический адрес ОП длиной 20 бит.

На рис. 2.4. проиллюстрировано формирование физического адреса ОП. В защищенном режиме базовые адреса сегментов хранятся в дескрипторных таблицах и имеют длину 24 или 32 бита (в зависимости от типа МП). В сегментных же регистрах хранится селектор, содержащий номер дескрипторной таблицы и дескрипторное смещение, т.е. порядковый номер дескриптора (в котором и хранится базовый адрес сегмента) в данной дескрипторной таблице.

Физический адрес очередной команды через внутреннюю магистраль МП и интерфейс памяти поступает на шину адреса системной магистрали. Одновременно из устройства управления (УУ) исполнительного блока на шину управления выдается команда (*управляющий сигнал*) в ОП, предписывающая выбрать число, находящееся по адресу, указанному в системной магистрали. Выбранное число, являющееся очередной командой, поступает из ОП через шину данных системной

магистрали, интерфейс памяти, внутреннюю магистраль МП на регистр команд (INST).

Из команды в регистре команд выделяется код операции, который поступает в УУ исполнительного блока для выработки управляющих сигналов, настраивающих микропроцессор на выполнение требуемой операции.

В зависимости от используемого в команде режима адресации организуется выборка необходимых исходных данных.

2.5. Поколения микропроцессоров семейства Intel

Родоначальником семейства микропроцессоров фирмы Intel является 16-и разрядный процессор 8086, выпущенный в июне 1978 года. Сегодня семейство 8086 насчитывает семь поколений процессоров.

Первое поколение (процессоры 8086 и 8088 и математический сопроцессор 8087) задало архитектурную основу – набор «неравноправных» 16-разрядных регистров, сегментную систему адресации в пределах 1 Мб с большим разнообразием режимов, систему команд, систему прерываний и ряд других атрибутов. В процессорах применялась «малая» конвейеризация: пока одни узлы выполняли текущую команду, блок предварительной выборки выбирал из памяти следующую.

Второе поколение (80286 и сопроцессор 80287) добавило в семейство так называемый «защищённый режим», позволяющий употреблять виртуальную память размером до 1Гб для каждой задачи, пользуясь адресуемой физической памятью в пределах 16 Мб. Защищённый режим является основой для построения многозадачных операционных систем, в которых система привилегий жестко регламентирует взаимоотношения задач с памятью, операционной системой и друг с другом. Производительность процессоров 80286 возросла не только в связи с ростом тактовой частоты, но и за счет значительного усовершенствования конвейера.

Третье поколение (80386/80387 с «суффиксами» DX и SX, определяющими разрядность внешней шины) ознаменовалось переходом к 32-разрядной архитектуре. Кроме расширения диапазона представляемых величин (16 бит отображают целые числа в диапазоне от 0 до 65535 или от –32768 до +32767, а 32 бита – более четырёх миллиардов), увеличилась ёмкость адресуемой памяти. На этих процессорах начала широко использоваться система Microsoft Windows.

Четвертое поколение (80486 также DX и SX) не внесло существенных изменений в архитектуру, зато был принят ряд мер для повышения производительности. В этих процессорах значительно усложнен исполнительный конвейер. В данном поколении отказались от внешнего сопроцессора – он стал размещаться на одном кристалле с центральным (либо его нет совсем).

Пятое поколение (процессор Pentium у фирмы Intel и K5 у фирмы AMD) дало суперскалярную архитектуру. Для быстрого снабжения конвейеров командами и данными из памяти шина данных этих процессоров сделана 64-разрядной, из-за чего их первое время иногда ошибочно называли 64-разрядными процессорами. «На закате» этого поколения появилось расширение MMX (Matrics Math Extensions {instruction set} – набор команд для расширения матричных

математических операций (первоначально Multimedia Extension {instruction set} – набор команд для мультимедиа-расширения)). Традиционные 32-разрядные процессоры способны выполнять сложение двух 8-разрядных чисел, размещая каждое из них в младших разрядах 32-разрядных регистров. При этом 24 старших разряда регистров не употребляются, и потому, например, при одной операции сложения ADD осуществляется просто сложение двух 8-разрядных чисел. Команды MMX оперируют сразу с 64 разрядами, где могут храниться восемь 8-разрядных чисел, причем имеется возможность выполнить их сложение с другими 8-разрядными числами в процессе одной операции ADD. Регистры MMX могут употребляться также для одновременного сложения четырех 16-разрядных слов или двух 32-разряных длинных слов. Такой принцип получил название SIMD (Single Instruction/Multiple Data - «один поток команд/много потоков данных») (2-4). Новые команды были предназначены в первую очередь для ускорения выполнения мультимедиа программ, но применять их можно не только к задачам, прямо связанным с технологией мультимедиа. В MMX появился и новый тип арифметики - с насыщением: если результат операции не помещается в разрядной сетке, то переполнения (или «антипереполнения») не происходит, а устанавливается максимально (или минимально) возможное значение числа.

Шестое поколение процессоров началось с Pentium Pro и продолжилось в процессорах Pentium III, Celeron и Xeon (у фирмы AMD сюда относятся процессоры K6, K6-2, K6-2+, K6-III). Ключевым здесь является динамическое исполнение, под которым понимается исполнение команд не в том порядке, как это предполагается программным кодом, а в том, как «удобно» процессору. Как пятое поколение по ходу развития было дополнено расширением MMX, так шестое поколение получило расширения, увеличивающие возможности MMX. У AMD это расширение 3dNow!, а у Intel - SSE (Streaming SIMD Extensions – потоковые расширения SIMD).

Седьмое поколение началось с процессора Athlon (у фирмы AMD). Причисление его к новому поколению обусловлено развитием суперскалярности и суперконвейерности. Седьмое поколение процессоров Intel началось позже с процессора Pentium 4 (осень 2000 года). Архитектура – «обобщенное определение системы с точки зрения существующих в ней информационных потоков и способа их обработки»

Конвейерная обработка предполагает разбивку выполнения каждой команды на несколько этапов, причём каждый этап выполняется на своей ступени конвейера процессора. При выполнении команда продвигается по процессору по мере освобождения следующих ступеней. Таким образом, на конвейере одновременно может обрабатываться несколько команд. Конвейер «классического» процессора Pentium имеет пять ступеней. Конвейеры процессоров с суперконвейерной архитектурой имеют большее количество ступеней, что позволяет упростить каждую из них и, следовательно, сократить время пребывания в них команд.

Скалярным называется процессор с одним конвейером. К этому типу относятся процессоры Intel до 80486-го включительно. Суперскалярный процессор имеет более одного конвейера (например, Pentium 2).

В ПЭВМ нашли применение не только микропроцессоры фирмы Intel. Крупнейшими производителями аналогов микропроцессорам Intel (клонов) являются фирмы Cyrix и AMD.

Фирма Cyrix выпускает микропроцессоры M-1 и M-2, аналогичные Pentium, но превосходящие его по производительности. Так, M-1 с тактовой частотой 150 МГц по производительности эквивалентен МП Pentium с тактовой частотой 200 МГц.

Фирма AMD, завоевавшая около 30% рынка МП в России, выпускает микропроцессоры К-5 и К-6, являющиеся соответственно аналогами Pentium и Pentium Pro.

P.S. См. презентации по теме из папки «Дополнительные материалы»

Лекция 3. Конвейерная организация и принципы конвейерной обработки

Основные вопросы:

3.1. Простейший конвейер команд и оценки его эффективности
3.2. Уровни конвейеризации
3.3. Понятие конфликтов в конвейере и пути их устранения: структурные конфликты, конфликты по данным и по управлению
3.4. Реализация точного прерывания в конвейере
3.5. Длинные конвейеры

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.

3.1. Простейший конвейер команд и оценки его эффективности

Для иллюстрации основных принципов построения процессоров будем использовать простейшую архитектуру, содержащую 32 целочисленных регистра общего назначения (*R*0,...,*R*31), 32 регистра плавающей точки (*F*0,...,*F*31) 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**.

Чтобы конвейеризовать выполнение команд можно просто разбить выполнение каждой команды на указанные выше этапы, отведя для выполнения каждого этапа один такт синхронизации, и начинать в каждом такте выполнение новой команды. Для хранения промежуточных результатов каждого этапа необходимо использовать регистровую станцию (память). Промежуточные регистровые станции 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. 3.1). Этот рисунок хорошо отражает совмещение во времени выполнения различных этапов команд. Однако чаще для представления работы конвейера используются временные диаграммы (рис. 3.2), 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 не конвейерной 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.

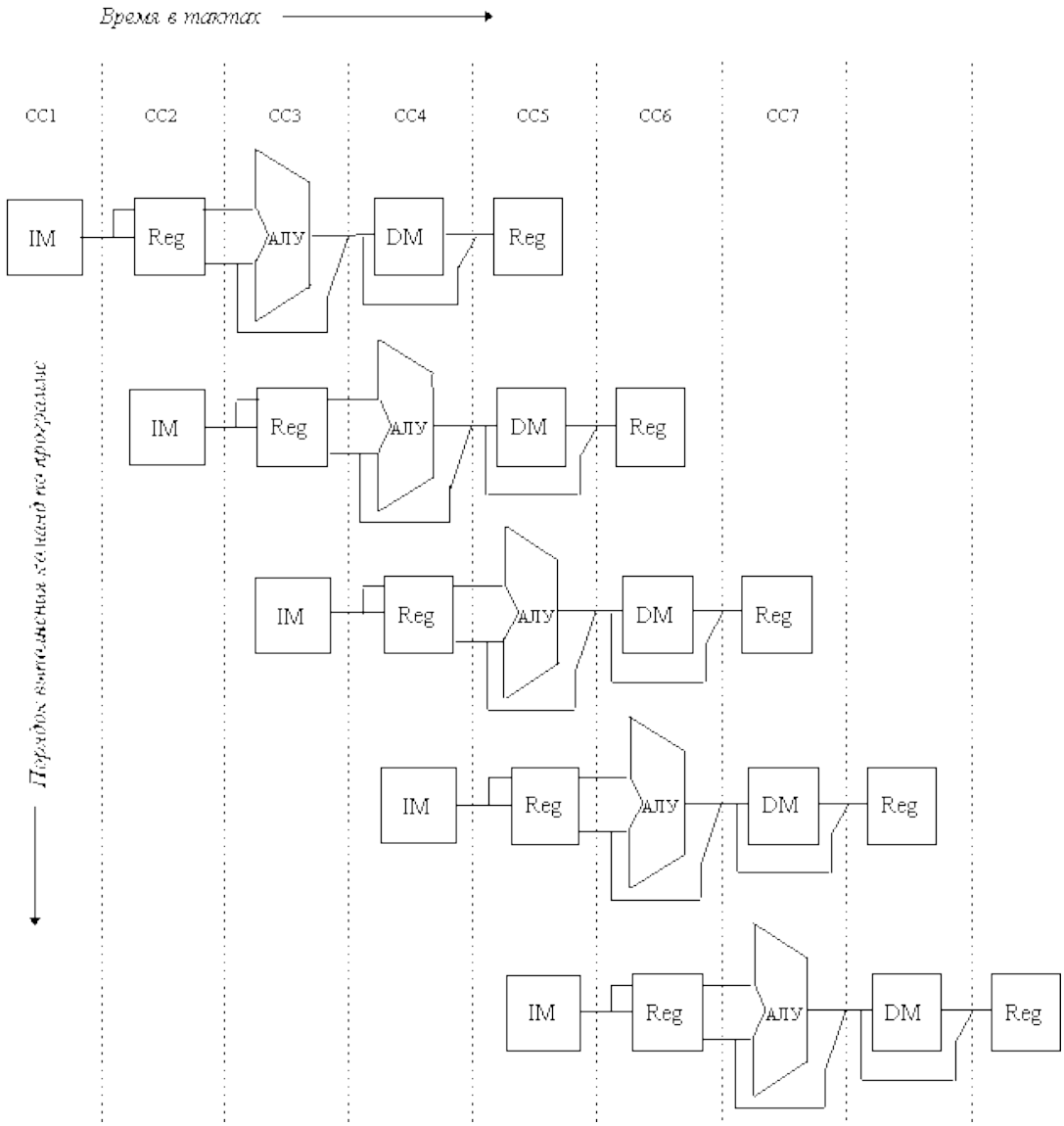


Рис. 3.1 Схемы процессора, реализующие конвейер команд

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. 3.2. Представление о работе конвейера в форме временной диаграммы

В качестве примера рассмотрим не конвейерную машину с пятью этапами выполнения операций, которые, например, имеют длительность 50, 50, 60, 50 и 50 нс соответственно (рис. 3.3). Пусть накладные расходы на организацию конвейерной обработки составляют 5 нс. Тогда среднее время выполнения команды в не конвейерной 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 отношению 260/65=4:

Среднее время выполнения команды в не конвейерном режиме = 260

Среднее время выполнения команды в конвейерном режиме = 6

$$Ускорение\ от\ конвейеризации\ команды = \frac{Среднее\ время\ вып.\ команды\ в\ неконвейерном\ режиме}{Среднее\ время\ вып.\ команды\ в\ конвейерном\ режиме}$$

Ускорение от конвейеризации = 260/65=4.

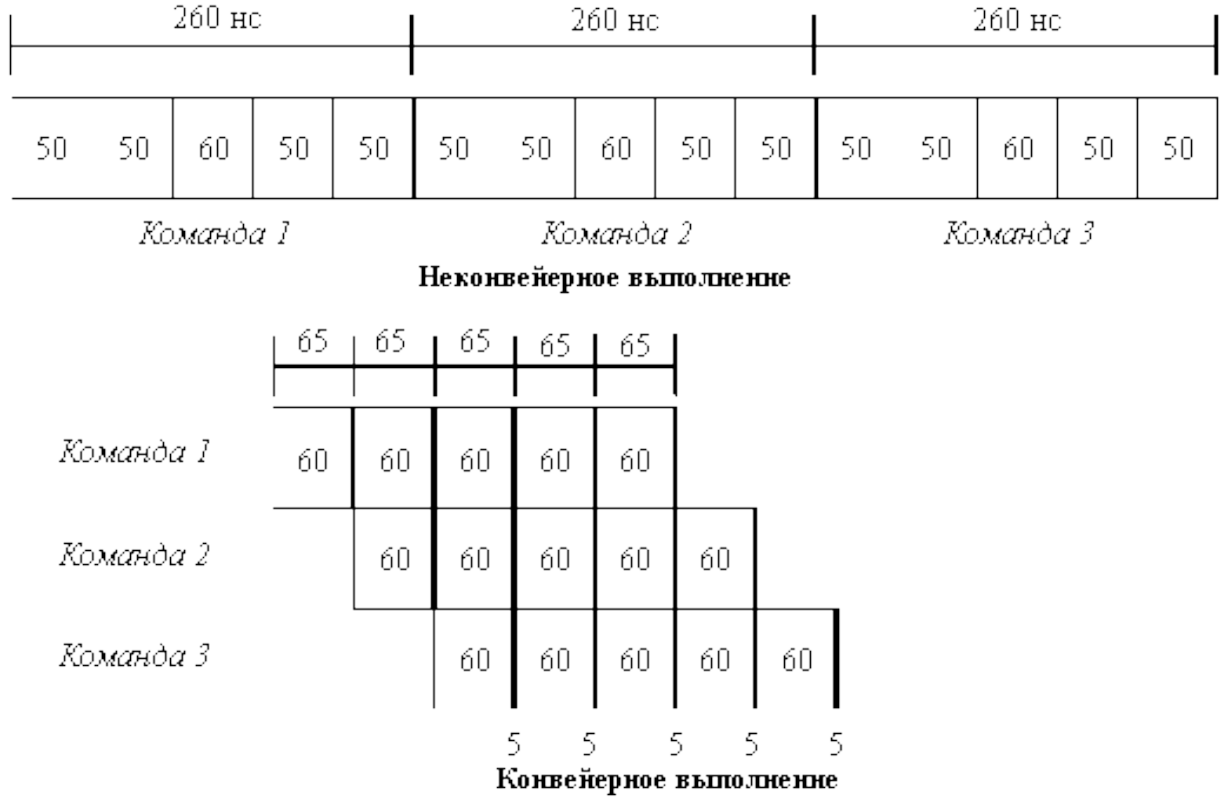


Рис. 3.3. Эффект от конвейеризации при выполнении 3-х команд – четырехкратное ускорение

Общая формула ускорения для арифметического конвейера может быть получена на примере сложения двух векторов длины n с плавающей запятой в конвейерном режиме и имеет вид:

$$\xi = \frac{n * k}{(n + k)}$$

где k – число ступеней конвейера, а n – длина одной последовательности исходных данных.

Важной характеристикой конвейерной обработки является среднее количество тактов (CPI) 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, минимизируем общий CPI конвейера и таким образом увеличиваем пропускную способность команд. Это выражение позволяет также охарактеризовать различные методы сокращения CPI, по тому компоненту общего CPI, который соответствующий метод уменьшает (см. ниже).

3.2. Уровни конвейеризации

- Макроконвейер – конвейеризация на уровне процессоров
- Конвейер команд – конвейеризация команд процессора
- Конвейер арифметический - конвейеризация на уровне выполнения команд процессора

3.3. Понятие конфликтов в конвейере и пути их устранения: структурные конфликты, конфликты по данным и по управлению

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.

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.

Рассмотрим различные типы конфликтов, возникающие при выполнении команд в конвейере, и способы их разрешения.

Существуют три класса конфликтов:

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. **Конфликты по данным**, возникающие в случае, когда выполнение одной команды зависит от результата выполнения предыдущей команды.
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.*

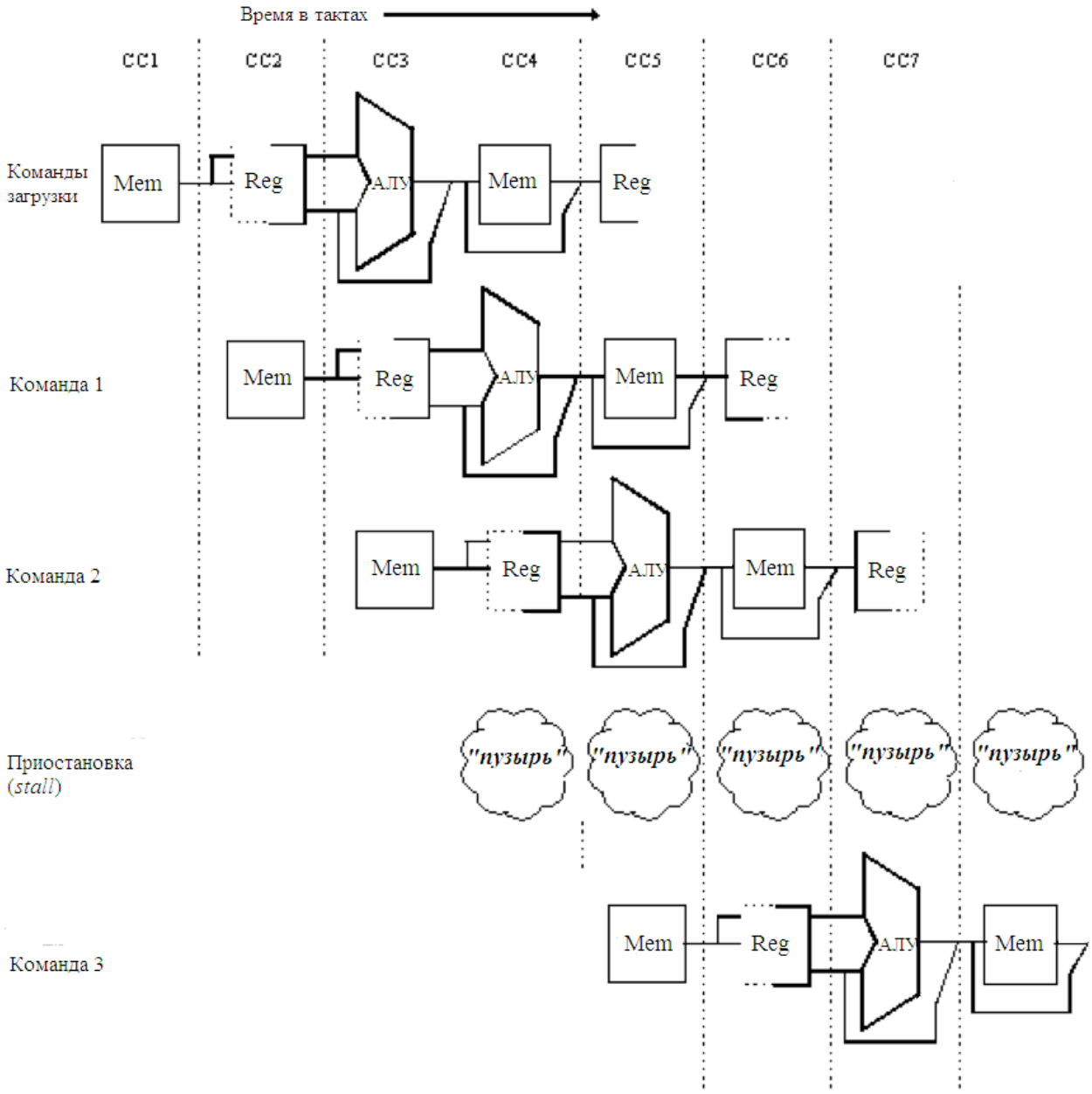


Рис. 3.4. Пример структурного конфликта при реализации памяти с одним портом

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.

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 не конвейерное, или не полностью конвейерное устройство, имеющее меньшую общую задержку, чем полностью конвейерное. Например, разработчики устройств обработки с плавающей точкой компьютеров CDC7600 и MIPS R2010 предпочли иметь меньшую задержку выполнения операций вместо полной их конвейеризации.

Одним из факторов, который оказывает существенное влияние на производительность конвейерных систем, являются межкомандные логические зависимости - ***конфликты по данным***. 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.

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	WB	
Команда 5							IF	ID	EX	WB
Команда 6								IF	ID	EX

Рис. 3.5. Диаграмма работы конвейера при структурном конфликте

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 не конвейерной машине. Рассмотрим конвейерное выполнение последовательности команд на рис. 3.6 и рис. 3.7.

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 мер для предотвращения

этого конфликта, то команда SUB прочитает неправильное значение и попытается его использовать.

ADD	R1, R2, R3	IF	ID	EX	MEM	WB				
SUB	R4, R4, R5		IF	ID	EX	MEM	WB			
AND	R6, R1, R7			IF	ID	EX	MEM	WB		
OR	R8, R1, R9				IF	ID	EX	MEM	WB	
OR	R10, R1, R11					IF	ID	EX	MEM	WB

Рис. 3.6. Последовательность команд в конвейере и ускоренная пересылка данных

ADD	R1, R2, R3	IF	ID	EX	MEM	WB				
			R			W				
SUB	R4, R4, R5		IF	ID	EX	MEM	WB			
				R			W			
AND	R6, R1, R7			IF	ID	EX	MEM	WB		
					R			W		
OR	R8, R1, R9				IF	ID	EX	MEM	WB	
						R			W	
OR	R10, R1, R11					IF	ID	EX	MEM	WB
							R			W

Рис. 3.7. Совмещение чтения и записи регистров в одном такте

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 (см. рис. 3.8).

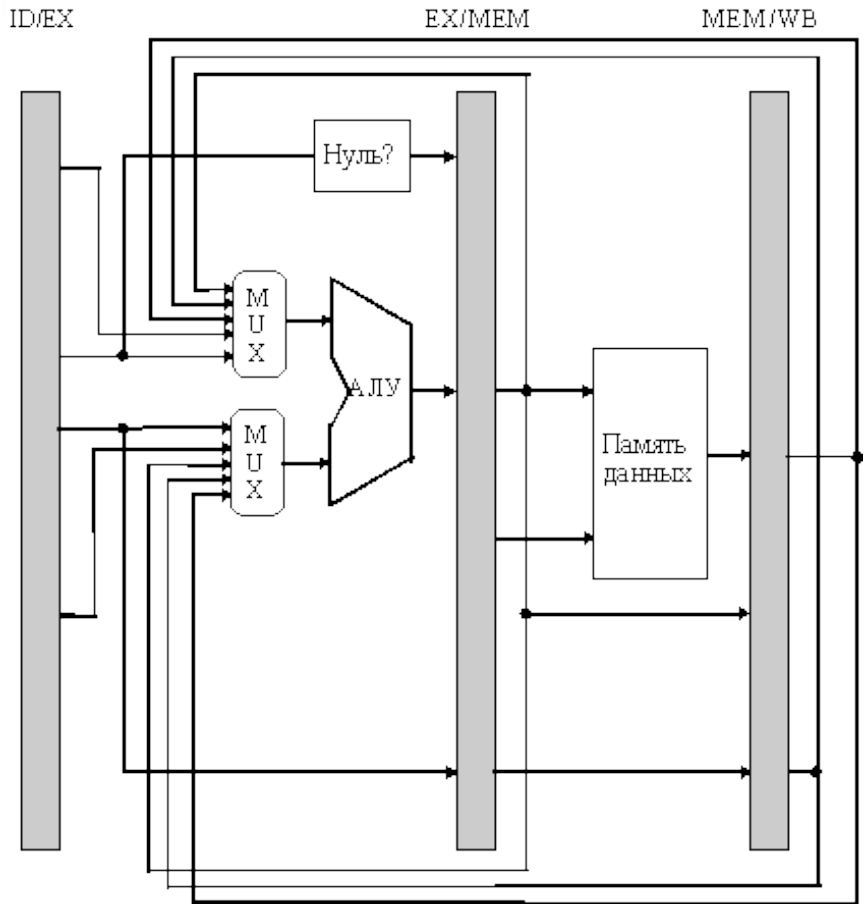


Рис. 3.8. АЛУ с цепями обхода и ускоренной пересылки

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

Конфликты по данным принято классифицировать на следующие три группы в зависимости от порядка операций чтения и записи. Рассмотрим две команды i и j, при этом i предшествует j.

- **RAW** (read after write) – 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** (write after read) – запись после чтения; j пытается записать результат в приемник прежде, чем он считывается оттуда командой i, так что i может некорректно получить новое значение. Особенно часто конфликты такого рода могут возникать в системах, допускающих выполнение команд не в порядке их расположения в программном коде.
- **WAW** (write after write) – 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).

К сожалению не все потенциальные конфликты по данным могут обрабатываться с помощью механизма "обходов", рассмотренного ранее. Пусть имеем следующую последовательность команд *с приостановкой конвейера*:

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

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:

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

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 не может улучшить ситуацию, однако в ряде более общих случаев возможно 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 получили наибольшее распространение.

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. Можно сгенерировать следующую последовательность команд:

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**

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.

Стратегия планирования на основе базовых блоков

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. **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. **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. **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).
4. **Метод переименования регистров** (register renaming). 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 до тех пор, пока не завершится выполнение соответствующей команды, значение в этом физическом регистре рассматривается как временное. 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 имеет дело только с логическими регистрами.

Метод переименования регистров упрощает контроль зависимостей по данным. В машине, которая может выполнять команды не в порядке их расположения в программе, номера логических регистров могут стать двусмысленными, поскольку один и тот же регистр может быть назначен последовательно для хранения различных значений. Но поскольку номера физических регистров уникально идентифицируют каждый результат, все неоднозначности устраняются.

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 (см. рис. 3.9).

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. 3.9. Приостановка конвейера при выполнении команды условного перехода

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 (**среднее число тактов на выдачу команды)**, 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.

Имеется несколько методов сокращения приостановок конвейера, возникающих из-за задержек выполнения условных переходов:

- метод выжидания;
- метод возврата;
- прогнозирование перехода как выполняемого;
- задержанные переходы.

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.3.9 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:**

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

Диаграмма работы модернизированного конвейера

Прогнозирование перехода как выполняемого

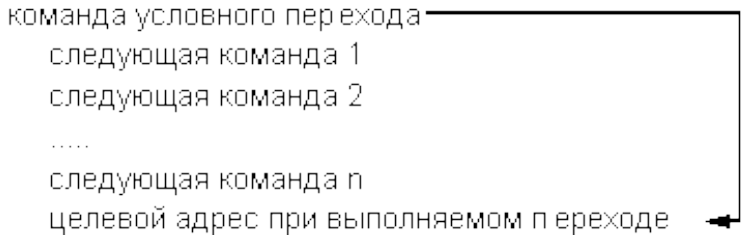
Как только команда условного перехода декодирована и вычислен целевой адрес перехода, предполагаем, что переход выполняемый, и осуществляем выборку команд и их выполнение, начиная с целевого адреса. Если мы не знаем целевой адрес перехода раньше, чем узнаем окончательное направление перехода, у этого 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:



Команды с 1 по *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. 3.10 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 (а) слот задержки заполняется независимой командой, находящейся перед командой условного

перехода. Это наилучший выбор. Стратегии (b) и (c) используются, если применение стратегии (a) невозможно.

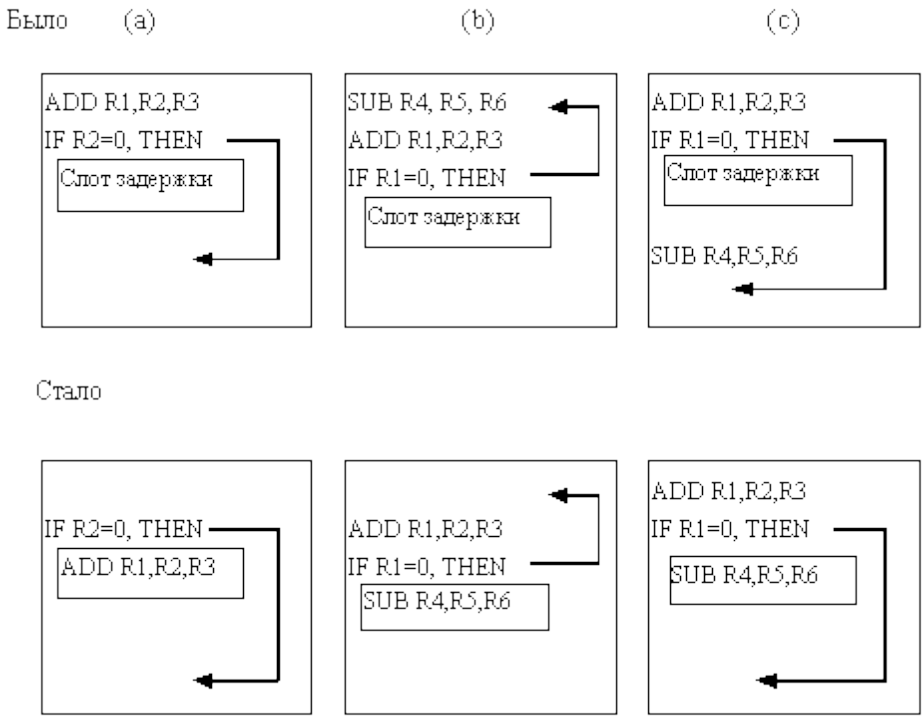


Рис. 3.10. Требования к переставляемым командам при планировании задержанного перехода

В последовательностях команд для случаев (b) и (c) использование содержимого регистра *R*1 в качестве условия перехода препятствует перемещению команды ADD (которая записывает результат в регистр *R*1) 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 предполагаем, что команда SUB выполнит ненужную работу, но вся программа при этом будет выполняться корректно. Это, например, может быть в случае, если регистр *R*4 используется только для временного хранения промежуточных результатов вычислений, когда переход выполняется не по прогнозируемому направлению.

В таблице указываются различные ограничения для всех приведенных выше 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**

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.

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).

3.4. Реализация 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.

Когда происходит прерывание, для корректного сохранения состояния машины необходимо выполнить следующие шаги:

1. В последовательность команд, поступающих на обработку в конвейер, *принудительно вставить команду перехода на прерывание*.
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. 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.***

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. 3.10 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. 3.10. Причины прерываний в простейшем конвейере

3.5. Длинные конвейеры

В рассмотренном нами конвейере стадия выполнения команды (**EX**) составляла всего один такт, что вполне приемлемо для целочисленных операций. Однако для большинства операций плавающей точки требуется большее число тактов. 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 ФУ, которое она использует, либо существует конфликт по данным.

Конвейер с такими дополнениями называется ***длинным конвейером***.

Рассмотрим понятие длинного конвейера на примере.

Допустим, что в нашей реализации процессора имеются четыре отдельных функциональных устройства (см. рис. 3.11):

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

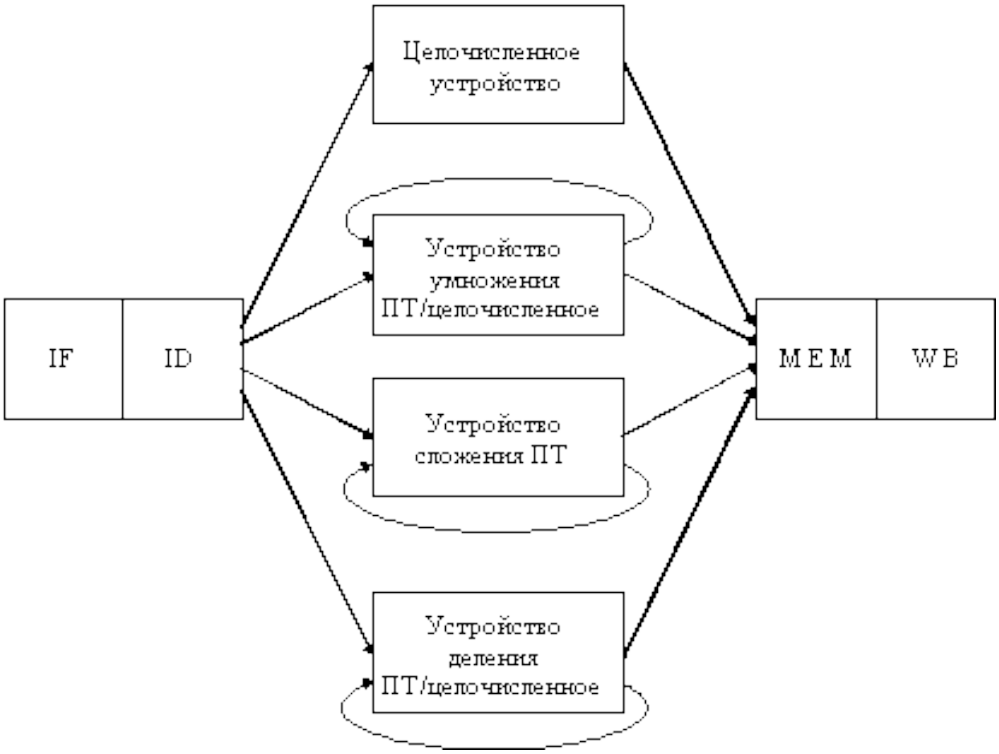


Рис. 3.11. Конвейер с дополнительными функциональными устройствами

Если предположить, что стадии выполнения других ФУ не конвейерные, то рис. 3.11 показывает структуру такого конвейера. Поскольку стадия **EX** является не конвейерной, никакая команда, использующая ФУ, 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.

В действительности промежуточные результаты, возможно, не используются циклически ступенью **EX**, как это показано на рис. 3.11, и ступень **EX** имеет задержки длительностью более одного такта. Можно обобщить структуру конвейера плавающей точки, допустив конвейеризацию некоторых ступеней и параллельное выполнение нескольких операций. Чтобы описать работу такого конвейера, необходимо определить задержки функциональных устройств, а также скорость инициаций или скорость повторения операций (скорость, с которой новые операции данного типа могут поступать в ФУ).

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

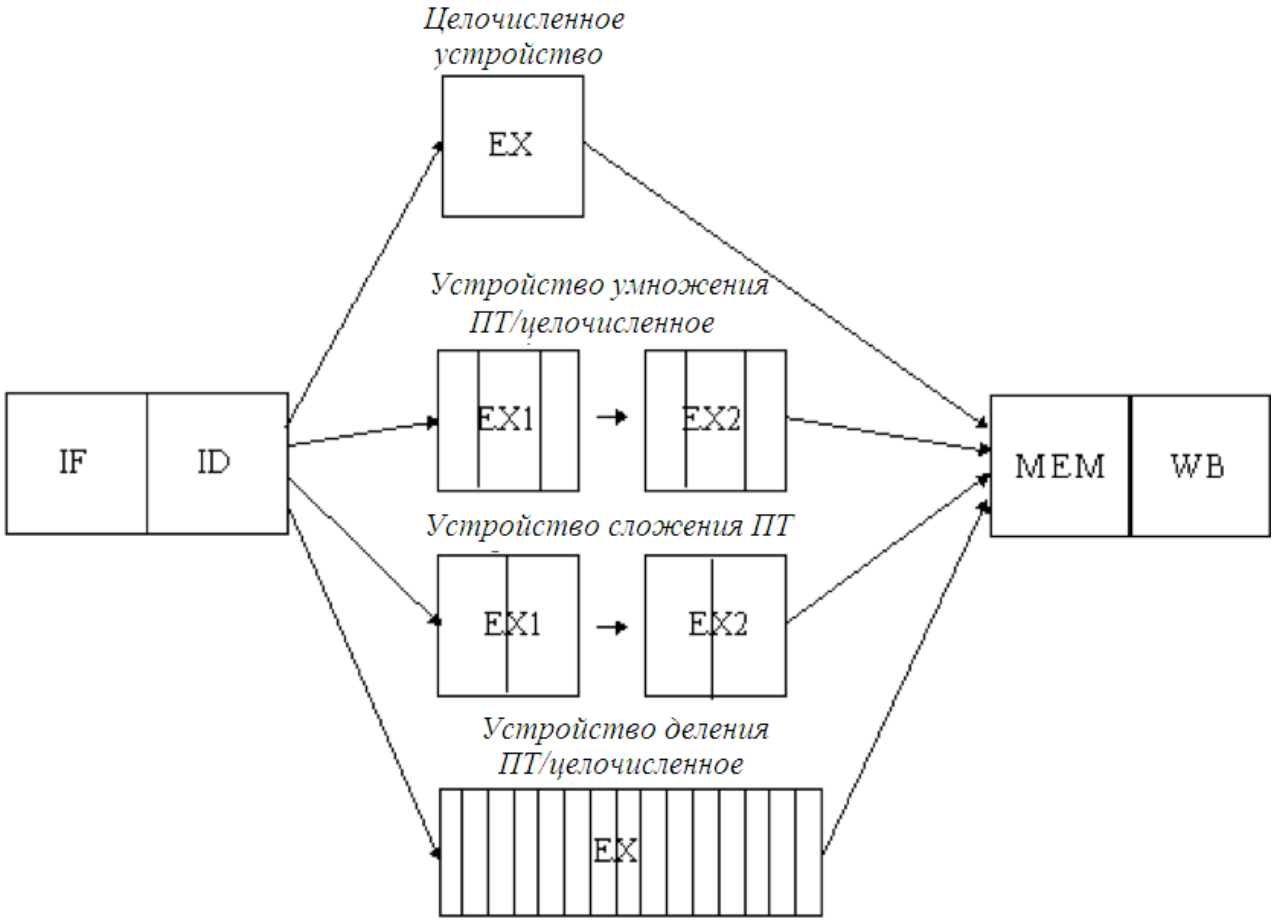


Рис. 3.12. Конвейер с многоступенчатыми функциональными устройствами

В заключение отметим несколько различных аспектов обработка конфликтов и организации ускоренных пересылок в длинных конвейерах:

1. Поскольку устройства не являются полностью конвейерными, в данной схеме возможны структурные конфликты. Эти ситуации необходимо обнаруживать и приостанавливать выдачу команд.
2. Поскольку устройства имеют разные времена выполнения, количество записей в регистровый файл в каждом такте может быть больше 1.
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. 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 плавающей точкой (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. 3.13, которая является **примером конфликта по записи в регистровый файл**.

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. 3.13. Пример конфликта по записи в регистровый файл

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.**

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. 3.13. Если бы команда **LD** была выдана на один такт раньше и имела в качестве месторасположения результата

регистр **F2**, то возник бы конфликт типа **WAW** (см. рис. 3.13.а), поскольку эта команда выполняла бы запись в регистр **F2** на один такт раньше команды **ADDD**.

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

Рис. 3.13.а. Пример конфликта **типа WAW**

Имеются два способа обработки этого конфликта типа **WAW**:

Первый подход заключается в задержке выдачи команды загрузки до момента передачи команды **ADDD** на ступень **MEM,** т.е. для нашего примера до такта **9**.

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. ***Проверка наличия структурных конфликтов.*** Ожидание освобождения функционального устройства и порта записи в регистры, если он потребуется.
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** (которая соответствует команде, выданной в предыдущем такте), **EX1/EX2** или **EX/MEM**.
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 плавающей точки создают также новые проблемы для механизма прерывания.

P.S. См. презентации по теме из папки «Дополнительные материалы»

Лекция 4. Векторные процессоры

Основные вопросы:

4.1. Введение

4.2. Особенности организации и функционирования

4.3. Архитектура векторно-конвейерного процессора на примере архитектуры суперкомпьютера Cray

4.4. Понятие векторизации циклов как эффективного инструментария при использовании векторных компьютеров

4.1. Введение

Современные высокопроизводительные вычислительные системы, в том числе супер-ЭВМ, имеют в своем составе несколько процессоров, чаще всего **двух типов**:

- **Скалярные**
- **Векторные**

Это связано с тем, что практически любую сложную задачу можно представить в виде совокупности двух частей: последовательной и параллельной.

Замечание. Отдельно выделяют суперскалярные процессоры. Суперскалярный процессор представляет собой нечто большее, чем обычный последовательный (скалярный) процессор. В отличие от последнего, он может выполнять несколько операций за один такт. Основными компонентами суперскалярного процессора являются устройства для интерпретации команд, снабженные логикой, позволяющей определить, являются ли команды независимыми, и достаточное число исполняющих устройств. В исполняющих устройствах могут быть конвейеры. Суперскалярные процессоры реализуют параллелизм на уровне команд.

Примером компьютера с суперскалярным процессором является IBM RISC/6000. Тактовая частота процессора у ЭВМ была 62.5 МГц, а быстродействие системы на вычислительных тестах достигало 104 Mflop (Mflop – единица измерения быстродействия процессора - миллион операций с плавающей точкой в секунду). Суперскалярный процессор не требует специальных векторизующих компиляторов, хотя компилятор должен в этом случае учитывать особенности архитектуры.

Последовательные части – совокупность однократно выполняемых разнотипных операций

Параллельная часть – блоки, выполняемые параллельно. Может быть явный параллелизм, если это независимые ветви программы, или обработка потоков данных, когда над многими элементами этих потоков выполняются одни и те же операции (обычно это циклы)

Реализация циклов на высокопроизводительных **скалярных процессорах** сопровождается рядом факторов, ограничивающих максимальную скорость вычислений:

1. Перед каждой скалярной операцией необходимо вызвать и декодировать скалярную команду.
2. Для каждой команды необходимо вычислить адреса данных
3. Данные должны вызываться из памяти, а результаты запоминаться в памяти. В условиях, когда каждая команда вырабатывает свой собственный запрос к памяти, возможны конфликты при обращении к памяти
4. Реализация команд построения циклов (счетчик, переход) сопровождается накладными расходами

Ограничить влияние факторов 1–3: применить конвейер команд и раздельные буферы команд и данных

Чтобы снять влияние всех факторов на производительность ВС – используют векторные процессоры.

4.2. Особенности организации и функционирования

Одним из самых простых и наиболее распространенных способов повышения быстродействия процессоров является конвейеризация процесса вычислений. ***Большим преимуществом конвейерных ЭВМ перед параллельными ЭВМ других типов является возможность использования пакетов программ, уже написанных для последовательных ЭВМ.***

Рассмотрим структуру и функционирование *скалярного конвейерного процессора* в целом (рис. 4.1).

Процессор содержит несколько конвейерных АЛУ. Это позволяет одновременно исполнять смежные арифметико-логические операции, что соответствует реализации не только параллелизма служебных операций, но и локального параллелизма. Для разных операций АЛУ имеют различную длину конвейера (на рис.4.1 длина равна 6, 7 и 14 позициям). В процессоре используются команды двух классов: команды обращения в память и регистровые команды для работы с РОН Буфер команд имеет многостраничную структуру, что позволяет во время работы УУ с одной страницей производить заранее смену других страниц.

Рассмотрим отрезок программы, соответствующий вычислению выражения:

$$Z = A - (B * C + D) - E.$$

На рис. 4.2 приведен информационный граф процесса выполнения выражения.

В программе: *LD* – команда загрузки операнда из памяти в регистр; *MP, SUB, ADD* – команды умножения, вычитания и сложения соответственно; *ST* – команда записи операнда из регистра в память.

Состояние некоторых регистров при выполнении программы показано в табл.4.1. В последней колонке таблицы приведен порядок запуска команд на исполнение. В частности, видно, что некоторые команды могут опережать по запуску команды, находящиеся в программе выше запущенной. Например, команды 4 и 5 выполняются ранее команды 3. Это возможно благодаря наличию в программе локального параллелизма и нескольких АЛУ в структуре процессора.

Однако подобные "обгоны" не должны нарушать логики исполнения программы, задаваемой ее информационным графом.

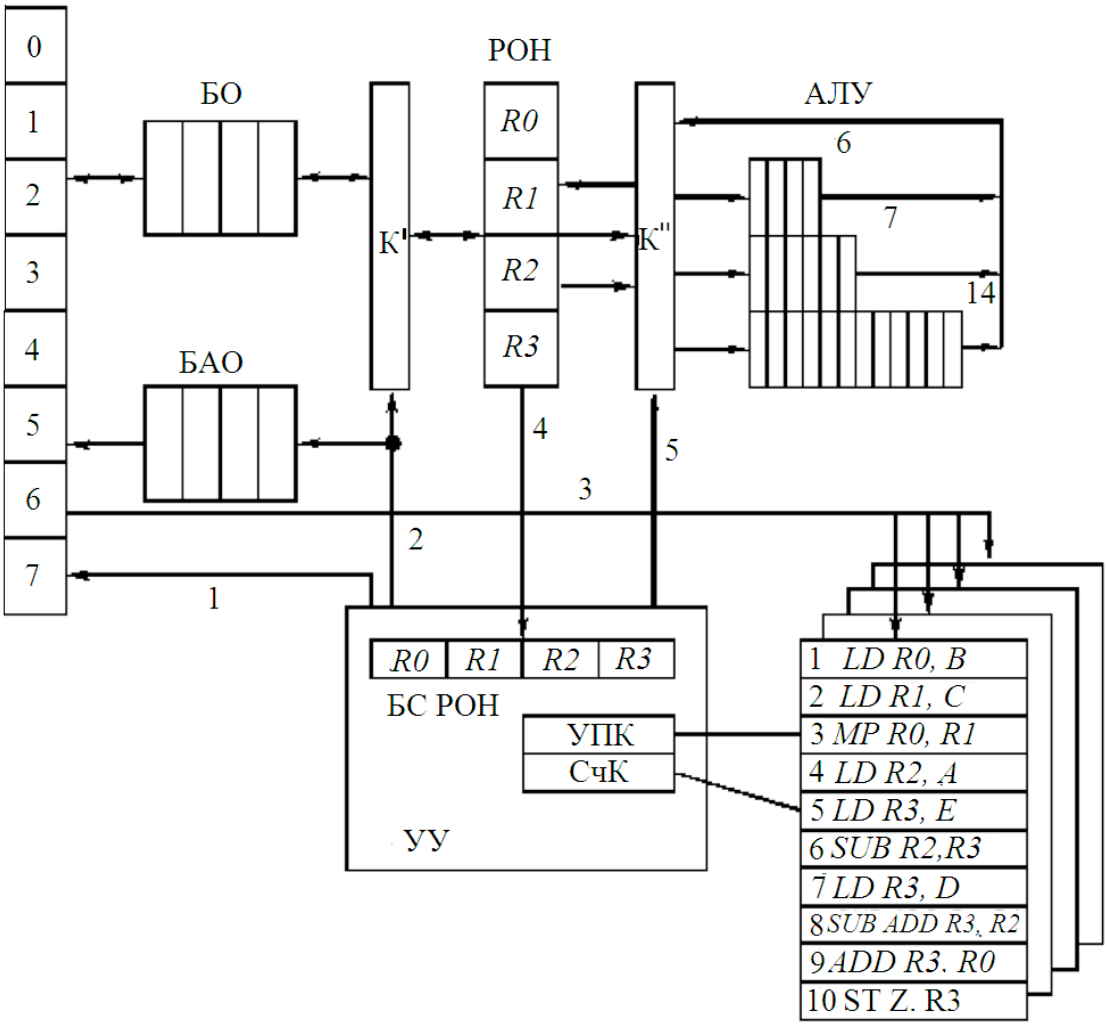


Рис. 4.1. Организация скалярного конвейерного процессора

ЦМП – центральная многоблочная память;
БАО – буфер адресов операндов;
АЛУ – конвейеризованные арифметико-логические устройства для сложения, умножения и деления чисел с плавающей запятой;
К', К" – коммутаторы памяти и АЛУ соответственно;
БС РОН – блок состояния РОН;
УПК – указатель номера пропущенной команды;
СчК – счетчик команд;
1 – шина адреса команд; 2 – шина управления выполнением команд обращения к памяти; 3 – шина заполнения БК; 4 – шина смены состояний РОН; 5 – шина управления выполнением регистровых команд

Любая операция согласно рисунку может быть запущена только после того, как подготовлены соответствующие операнды. Это достигается путем запрета доступа в определенные РОН до окончания операции, в которой участвуют данные РОН. Состояния РОН отражены в специальном блоке состояний (БС) РОН.

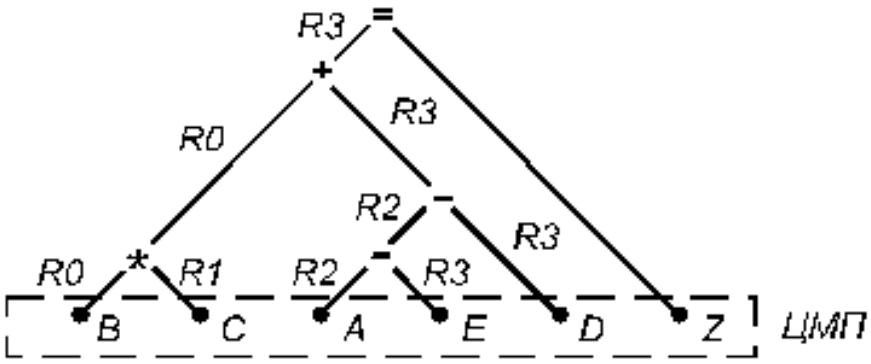


Рис. 4.2. Информационный граф программы: ***Ri*** – регистр общего назначения

В табл. 4.1 приведено также описание нескольких тактов работы процессора. Принято, что выборка операнда из ЦМП занимает четыре такта. Кроме того, считается, что за один такт процессора устройство управления запускает на исполнение одну команду или просматривает в программе до четырех команд.

Таблица 4.1.

Порядок исполнения программы в скалярном конвейере

NN такта	Состояние РОН				Номер команды
	*R*0	*R*1	*R*2	*R*3	
1	4	–	–	–	1
2	3	4	–	–	2
3	2	3	4	–	4
4	1	2	3	4	5
5	x	1	2	3	–
6	7	–	1	2	3
7	6	–	x	1	–
8	5	–	6	x	6
9	4	–	5	4	7
10	3	–	4	3	–

Пояснения к таблице.

Такт 1. Анализ БС РОН показывает, что все РОН свободны, поэтому команда 1 запускается для исполнения в ЦМП. В столбец *R*0 записывается 4, что означает: *R*0 будет занят четыре такта. После исполнения каждого такта эта величина уменьшается на единицу. В структуре процессора занятость *R*0 описывается установкой разряда *R*0 БС РОН в 1, а затем сброс *R*0 в 0 по сигналу с шины 4, который появляется в такте 4 после получения операнда из памяти.

Такт 2. Запускается команда 2 и блокируется регистр *R*1.

Такт 3. Просматривается команда 3, она не может быть выполнена, так как после анализа БС РОН нужные для ее исполнения регистры *R*0 и *R*1 заблокированы. Команда 3 пропускается, а ее номер записывается в УПК. Производится анализ

условий запуска следующей (по состоянию СчК) команды. Команда 4 может быть запущена и запускается.

Такт 4. Просмотр БК начинается с номера команды, записанной в УПК. Команда 3 не может быть запущена, поэтому запускается команда 5.

Такт 5. Команда 3 не может быть запущена, так как занят регистр *R*1, однако регистр *R*0 освободился и может использоваться командой 3, но он снова блокируется (символ **x**). Просмотр четырех следующих команд показывает, что они не могут быть запущены, поэтому в такте 5 для исполнения выбирается новая команда.

Такт 6. Запускается команда 3.

В дальнейшем процесс происходит аналогично. Можно заметить, что за 10 тактов, описанных в табл. 1, в процессоре запущено семь команд, что соответствует 10/7 = 1,5 такта на команду. Предположим, что такт процессора равен 10 нс. Тогда на выполнение одной команды тратится 15 нс, что соответствует быстродействию $V = 70$ млн оп/с.

Будем считать, что вектор – это одномерный массив, который образуется из многомерного массива, если один из индексов не фиксирован и пробегает все значения в диапазоне его изменения. В некоторых задачах векторная форма параллелизма представлена естественным образом. В частности, рассмотрим задачу перемножения матриц.

Для перемножения матриц на последовательных ЭВМ неизменно применяется гнездо из трех циклов:

```
   DO 1 I = 1, L
   DO 1 J = 1, L
   DO 1 K = 1, L
1 C(I, J) = C(I, J) + A(I, K) * B(K, J)
```

Внутренний цикл может быть записан в виде отрезка программы на фортраноподобном параллельном языке:

```
R (*) = A (I, *) * B (*, J)          (*)
C (I, J) = SUM R (*)
```

Здесь ***R(*), A(I, *), B(*, J)*** - векторы размерности *L*; первый оператор представляет бинарную операцию над векторами, а второй – унарную операцию SUM суммирования элементов вектора.

Для выполнения операций над векторами также используются арифметические конвейеры.

Структура конвейерного процессора для обработки векторов показана на рис. 4.3.

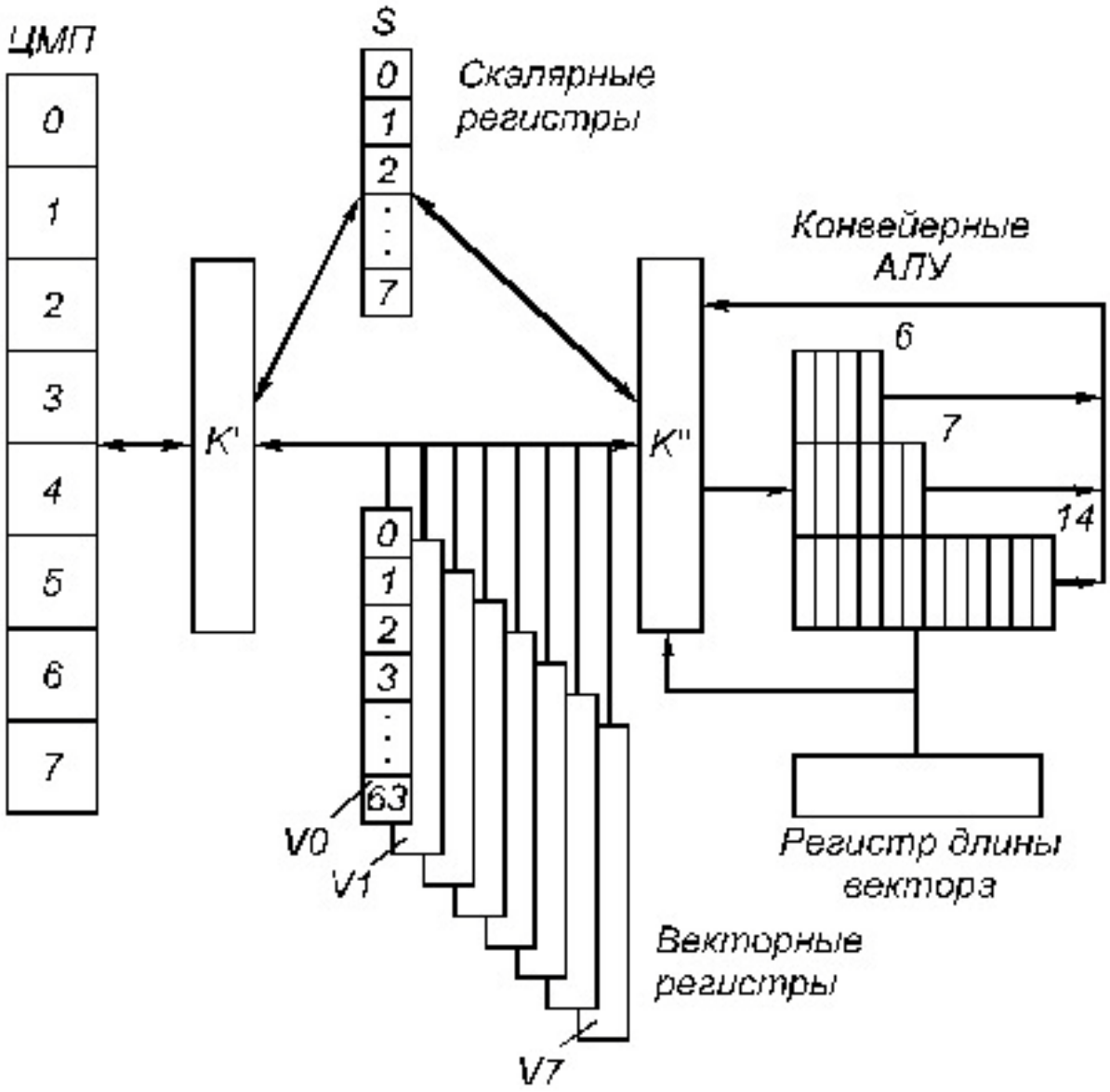


Рис. 4.3. Структура векторного конвейерного процессора

Структура устройства управления векторного процессора аналогична структуре УУ скалярного конвейерного процессора. Отличие лишь в том, что при выполнении векторной команды код операции команды не меняется.

Главная особенность векторного процессора - наличие ряда *векторных регистров V* (обычно до 8), каждый из которых позволяет хранить вектор длиною до 64 слов. Это своего рода РОН, значительно ускоряющие работу векторного процессора. Назначение остальных узлов такое же, как и узлов, изображенных на рис. 4.2.

Рассмотрим, как будет выполняться программа (*) в векторном процессоре. На условном языке ассемблерного уровня программа может быть представлена следующим образом:

1 LD L, Vi, A
2 LD L, Vj, B
3 MP Vk, Vi, Vj (**)

4 SUM Sn, Vk

Операторы 1 и 2 соответствуют загрузке слов из памяти с начальными адресами A и B в регистры *Vi, Vj*; оператор 3 означает поэлементное умножение векторов с размещением результата в регистре *Vk*; оператор 4 – суммирование вектора из *Vk* с размещением результата в *Sn*.

Соответственно этой программе, векторные регистры сначала по-тактно заполняются из ЦМП, а затем слова из векторных регистров по-тактно (одна пара слов за такт) передаются в конвейерные АЛУ, где за каждый такт получается один результат.

Рассмотрим характеристики быстродействия векторного процессора на примере выполнения команды *MP Vi, Vj, Vk*.

Число тактов, необходимое для выполнения команды, равно:

$r = m_* + L$, где m_* – длина конвейера умножения.

Поскольку на умножение пары операндов затрачивается $k = (m_* + L)/L$ тактов, то быстродействие такого процессора

$$V = 1/(K\Delta t) = \frac{L}{(m_* + L)\Delta t},$$

где Δt – время одного такта работы конвейера.

Быстродействие конвейера зависит от величины L (рис. 4.4, кривая 1).

При $L > m_*$ величина $K \sim 1$ и $V \sim 1/\Delta t$. Обычно для векторных процессоров стараются сделать Δt малым, в пределах 10...20 нс, поэтому быстродействие при выполнении векторных операций может достигать $50...10*10^6$ оп/с.

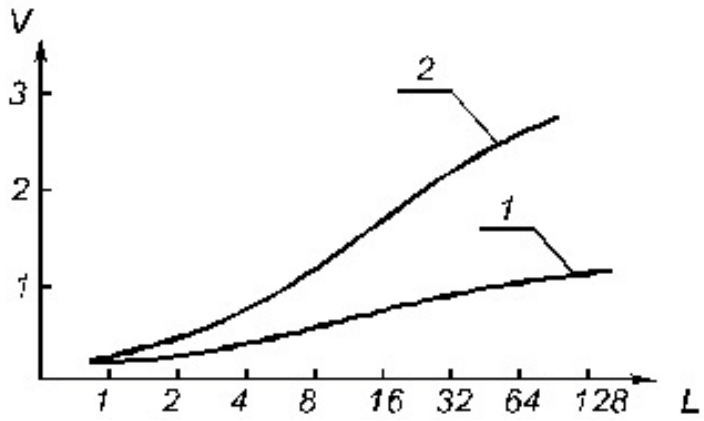


Рис. 4.4. Зависимость быстродействия процессора от длины вектора: $m_{ЦМП} = 4$; $m_* = 7$; $m_+ = 6$

Важной особенностью векторных конвейерных процессоров, используемой для ускорения вычислений, является механизм *зацепления*. Зацепление – такой способ вычислений, когда одновременно над векторами выполняется несколько операций. В частности, в программе можно одновременно производить выборку

вектора из ЦМП, умножение векторов, суммирование элементов вектора. Поэтому программу можно переписать следующим образом:

LD L, Vi, A

ЗЦ Sn, Vi, B

Здесь команда зацепления (ЗЦ) задает одновременное выполнение операций в соответствии со схемой соединений (рис. 4.5).

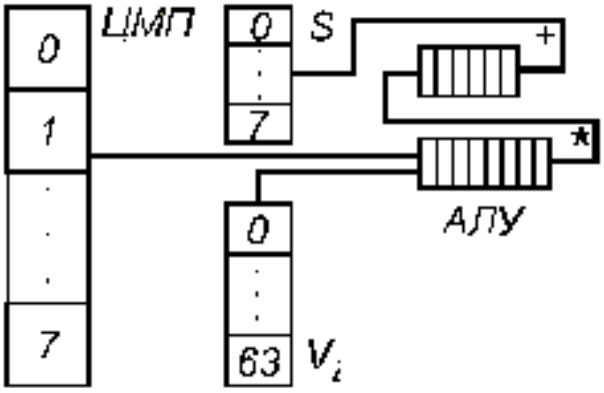


Рис. 4.5. Выполнение операции зацепления в векторном конвейерном процессоре

Для команды ***ЗЦ*** получаем:

$$r = m_{ЦМП} + m_{*} + m_{+} + L,$$

$$K = (m_{ЦМП} + m_{*} + m_{+} + L) / L,$$

$$V = n / (K\Delta t),$$

где n – число одновременно выполняемых операций. В случае команды ***ЗЦ*** для $n = 3$ быстродействие процессора возрастает (рис.4.4, кривая 2).

При $L >> m_i$ и Δt = 10...20 нс в зацеплении быстродействие равно 150...300 млн оп/с. Такое быстродействие достигается не на всех векторных операциях.

Для векторных ЭВМ существуют "неудобные" операции, в которых ход дальнейших вычислений определяется в зависимости от результата каждой очередной элементарной операции над одним или парой операндов. В подобных случаях L приближается к единице.

К таким операциям относятся операции рассылки и сбора, которые можно определить следующими отрезками фортран-программ:

1) рассылка

```
   DO 1 I = 1, L
1  X(INDEX (I)) = Y (I)
```

2) сбор

```
   DO 1 I = 1, L
1 Y (I) = X (INDEX (I))
```

Целочисленный массив INDEX содержит адреса операндов, разбросанных произвольным образом в памяти процессоров. Операция рассылки распределяет упорядоченный набор элементов *Y(I)* по всей памяти в соответствии с комбинацией адресов в массиве INDEX. Операция сбора, наоборот, собирает разбросанные элементы *X* в упорядоченный массив *Y*. Названные операции имеются в задачах сортировки, быстрого преобразования Фурье, при обработке графов, представленных в форме списка, и во многих других задачах.

Быстродействие векторного процессора на таких операциях снижается до уровня быстродействия скалярного процессора.

Выше конвейерные ЭВМ были описаны по частям: сначала скалярная часть, затем векторная.

4.3. Архитектура векторно-конвейерного процессора на примере архитектуры суперкомпьютера Cray

Архитектура типичного векторного суперкомпьютера Cray следующая (в качестве примера рассмотрим старую модель Cray-1):

- Процессор имеет 8 векторных регистров, каждый из которых может хранить 64 слова по 64 бита каждое.
- Есть также 8 скалярных регистров для 64-битовых слов и
- 8 регистров для адресации для 20-битовых слов.
- Вместо кэш-памяти используются специальные регистры, управление которыми осуществляется программным путем из выполняющейся программы.

Всего компьютер Cray-1 содержал до 12 конвейеризованных процессоров, имевших отдельные конвейеры для различных арифметических и логических операций. Скорость работы процессора строго согласована со скоростью работы оперативной памяти.

CRAY-1, созданная в 1976 г., была одной из первых ЭВМ, в которой в полной мере проявились все характерные особенности *векторно-конвейерных машин*. Машина CRAY-1 включает стандартные для любой ЭВМ секции: управления памятью и ввода-вывода, регистровую секцию и группу функциональных устройств. Однако состав оборудования каждой секции существенно отличается от аналогичных устройств последовательных ЭВМ.

Память используется как для выборки команд и векторных данных (конвейерный режим), так и для выборки скалярных данных (поточный режим). Для организации поточного режима применяются буферные регистры адреса (*B*0...*B*63) и данных (*T*0...*T*63). Память обладает большой пропускной способностью, однако для разных узлов не всегда используется полная пропускная способность. Максимальная скорость нужна для заполнения буферов команд.

Структура УУ: здесь имеются механизмы предварительного просмотра команд, а также резервирования регистров и функциональных устройств.

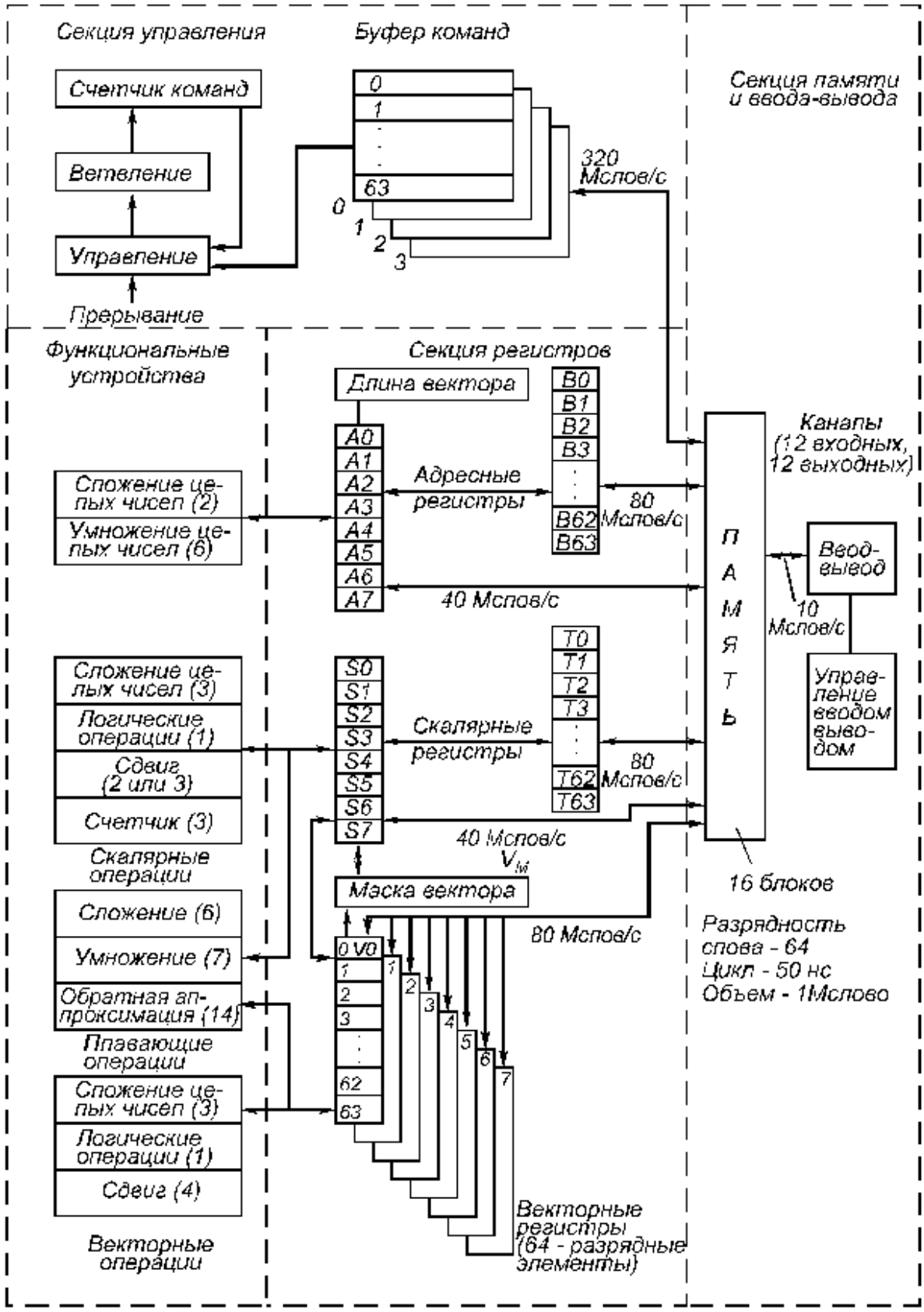


Рис. 4.6. Архитектура процессора CRAY-1.

Для выполнения адресных операций используются группа адресных регистров *A*0...*A*7 и специальные АЛУ для операций целочисленной арифметики.

Эти АЛУ, как и все другие функциональные устройства, имеют конвейерную структуру.

Скалярные операции выполняются с помощью группы регистров *S*0...*S*7, а векторные – с помощью векторных регистров *V*0...*V*7.

Наибольший интерес представляет состав функциональных устройств. В выполнении скалярных операций участвует семь функциональных устройств: четыре для работы с целочисленной арифметикой и три для работы с числами с плавающей запятой. Устройство "Счетчик" используется для подсчета числа единиц в операнде или числа нулей, предшествующих первой единице операнда.

Поскольку деление плохо поддается конвейеризации, в CRAY-1 оно выполняется в устройствах обратной аппроксимации и умножения посредством итерационной процедуры. Такой подход позволяет использовать конвейерную обработку на операции умножения и зацепление операций.

Векторные операции в CRAY-1 можно разделить на четыре типа. Векторная инструкция первого типа получает операнды из одного или двух регистров *V* и отправляет результат в другой регистр *V*. Последовательные пары операндов передаются из *Vj* и *Vk* в конвейерное АЛУ в каждом такте, и соответствующий результат на выходе АЛУ появляется на *m* тактов позже, где *m* – длина конвейера. Результат отправляется в регистр результата *Vi*. Векторная инструкция второго типа получает по одному операнду из регистров *S* и *V*. Инструкции двух других типов передают данные между памятью и регистрами *V*.

При выдаче векторной инструкции требуемое АЛУ и регистры операндов резервируются на число тактов, определяемое длиной вектора. Последующая векторная инструкция, требующая тех же ресурсов, что и выполняемая, не может выполняться до тех пор, пока ресурсы не будут освобождены. Однако выполнение последующих инструкций, не пересекающихся по ресурсам с неоконченной инструкцией, разрешается.

Две команды машины CRAY-1 требуют специального объяснения. Установка маски 64-разрядного векторного регистра маски (*VM*) соответствует 64 элементам векторного регистра. Если элемент удовлетворяет условию, то соответствующий разряд *VM* устанавливается в 1, в противном случае – в 0. Таким образом, команда, *VM V5, Z* устанавливает разряд *VM* в 1, когда элементы *V5* равны нулю; *VM V7, P* устанавливает разряд *VM* в 1, если элементы *V7* положительны.

По команде слияния векторов содержимое двух векторных регистров *Vi* и *Vk* сливается в один результирующий вектор *Vi* в соответствии с маской регистра *VM*. Если *l*-й разряд *VM* – единица, то *l*-й элемент *Vj* становится *l*-м элементом регистра результатов, в противном случае *l*-й элемент *Vk* становится *l*-м элементом регистра результатов. Значение регистра длины вектора определяет число сливаемых элементов. Таким образом, команда *ViVj!Vk&VM* сливает *Vj* и *Vk* в *Vi* в соответствии с комбинацией в *VM*; *V7S2!V6&VM* сливает *S2* и *V6* в *V7* в соответствии с комбинацией в *VM*.

Цель команд маски и слияния – обеспечить условные вычисления с векторными командами.

В CRAY-1 соотношение объема памяти хорошо сбалансировано с производительностью системы. По эмпирическому правилу Амдала 1 бит памяти должен приходиться на 1 оп/с. В CRAY-1 при памяти 1 *М*слов (64 Мбит) производительность равна $80*10^6$ оп/с.

В данной системе впервые применено зацепление операций. За счет этого могут работать два, а иногда три функциональных устройства, доводя производительность системы до $160*10^6$ и даже $240*10^6$ результатов с плавающей запятой в секунду.

4.4. Понятие векторизации циклов как эффективного инструментария при использовании векторных компьютеров

Для работы на векторных ЭВМ наиболее удобными являются ***массивы с длиной, равной длине векторного регистра***. Однако это благое пожелание очень редко выполняется. Более того, часто число элементов массива вообще не кратно 128. Рассмотрим простейший цикл, который можно векторизовать.

Пусть дан массив *m* длины 128, который надо заполнить по следующему алгоритму:

```
do i=1, 128
m(i) = i
enddo
```

Воспользуемся командами ассемблера несуществующей ЭВМ, но вполне отражающими смысл операций с векторными регистрами. Приведенный цикл можно записать на ассемблере так:

```
SETLEN #128 ; установить используемое число
            ; элементов в векторных регистрах (во всех)
SETINC #4   ; установить смещение к последующему
            ; элементу массива в памяти (4 байта)
SETNUM v0   ; записать в элементы векторного регистра v0
            ; их номера начиная с нуля и кончая 127
ADD #1, v0  ; добавить 1 к каждому элементу регистра v0
SAVE v0, m  ; записать элементы v0 в ОЗУ в последовательные
            ; слова (смещение = 4) начиная с адреса m
```

Обратите внимание на 3 команду – каждый элемент векторного регистра "знает" свой номер. Это делает очень простым вычисление переменной цикла: после добавления 1 значение переменной получается записанным в соответствующий элемент вектора. Всего 5 последовательных команд векторного процессора выполняют цикл из 128 повторений. Здесь нет ни команд сравнения, ни условных переходов.

Теперь увеличим размер массива и число повторений цикла до *N*:

```
do i=1, N
  m(i) = i
enddo
```

Для правильной работы процессора необходимо установить число используемых элементов вектора не более, чем 128. Если N будет произвольным, то придется превратить данный одинарный цикл в двойной:

```
1   do inc=0, N-1, 128
2   NN = min0(128, N-inc)
3   do i=1, NN
      m(inc+i) = inc+i
      enddo
      enddo
```

Цикл с меткой 1 выполняет "разбиение" массива на подмассивы длиной 128 элементов. Переменная *inc* имеет смысл смещения от первого элемента массива к очередному подмассиву: 0, 128, 256... Переменная *NN* определяет длину подмассива. Обычно она равна 128, но последний подмассив может иметь меньшую длину, если *N* не кратно 128. Функция *min*0 выбора минимального значения в операторе с меткой 2 выдает значение не 128 только для последнего подмассива. Внутренний цикл с меткой 3 практически эквивалентен циклу из предыдущего примера.

Имеется только 3 отличия:

- число элементов в векторном регистре равно *NN*;
- к параметру цикла дополнительно надо добавлять значение *inc*;
- запись элементов вектора в память осуществляется начиная не с начала массива, а с элемента с номером *i+inc*.

Этот цикл может быть записан в машинных командах примерно так:

```
SETLEN NN      ; число элементов в векторных регистрах = NN
SETINC #4      ; смещение к последующему элементу массива
SETNUM v0      ; записать в элементы векторного регистра v0
               ; их номера начиная с нуля и кончая 127
ADD #1, v0     ; добавить 1 к каждому элементу регистра v0
ADD inc, v0    ; добавить значение переменной inc
MOVE inc, r0   ; записать в скалярный регистр r0 значение
               ; переменной inc - смещение от первого
SETLEN NN      ; элемента массива к m(inc+1)
MUL #4, r0     ; умножить на 4 - смещение в байтах
ADD #m, r0     ; добавить адрес массива m, получается адрес
               ; элемента m(inc+1)
SAVE v0, @r0   ; записать элементы v0 в ОЗУ в последовательные
               ; слова (смещение = 4) начиная с адрес
               ; хранящегося в регистре r0
```

По отношению к простейшему циклу добавились **одна векторная и три скалярных** команды. Однако это только внутренний цикл. Охватывающий его цикл с меткой 1 и вычисление *NN* создадут дополнительный код, который будет выполняться столько же раз, сколько и код для внутреннего цикла. Транслятор всегда будет создавать охватывающий цикл, если *N* есть переменная, а не константа со значением от 1 до 128 (для 128-элементного векторного регистра).

Из сказанного выше следует, что ***эффективность векторной программы будет невысокой при работе с небольшими массивами и длина которых заранее неизвестна.***

Кратные циклы (например, с тремя повторениями) могут в векторном режиме выполняться медленнее, чем в скалярном на той же машине.

Пути повышения эффективности функционирования конвейера (см. дополнительный файл)

P.S. См. презентации по теме из папки «Дополнительные материалы»

Лекция 5. Организации памяти ЭВМ и систем

Основные вопросы:

5.1. Введение
5.2. Системы памяти – критерии оценки.
5.3. Основные характеристики современных запоминающих устройств (ЗУ). Классификация ЗУ
5.4. Полупроводниковые запоминающие устройства: организация ЗУ с произвольным доступом
5.5. Постоянные запоминающие устройства: разновидности
5.6. Регистровая память

5.1. Введение

Память представляет собой одну из важнейших подсистем ЭВМ, во многом определяющую их производительность. Тем не менее, в течение всей истории развития вычислительных машин она традиционно считается их "*узким местом*".

Память, запоминающие устройства (ЗУ) (англ. memory, storage) – среда или функциональная часть ЭВМ, предназначенная для приема, хранения и избирательной выдачи данных.

Системы памяти (СП) современных ЭВМ представляют собой **совокупность аппаратных средств**, предназначенных для хранения используемой в ЭВМ информации. На рис. 5.1 представлен возможный состав системы памяти.

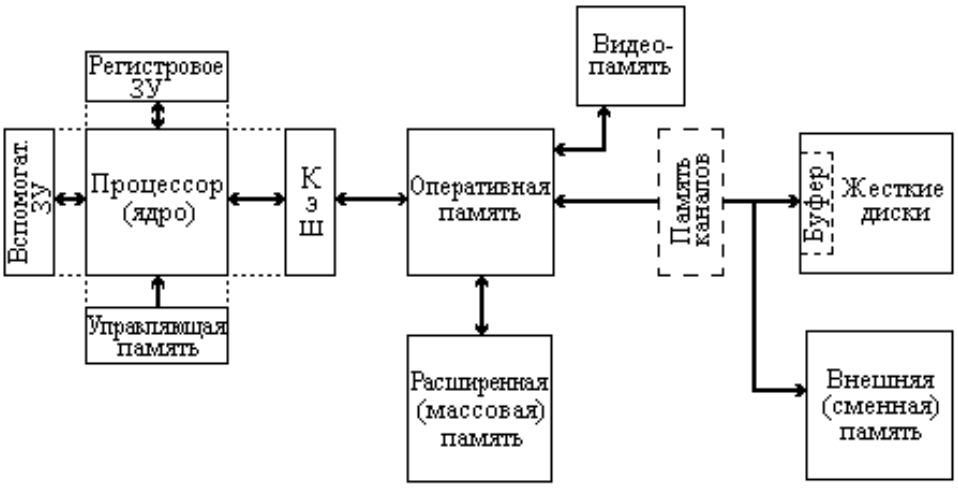


Рис. 5.1. Возможный состав системы памяти ЭВМ

К хранимой в СП информации относятся обрабатываемые данные, прикладные программы, системное программное обеспечение и служебная информация различного назначения. **К системе памяти можно отнести и программные средства**, организующие **управление** ее работой в целом, а также **драйверы** различных видов запоминающих устройств.

5.2. Системы памяти – критерии оценки

Рис. 5.2 дает представление о группах факторов, влияющих на работоспособность ЗУ, которые определяют информационную и конструктивную надежность и эффективность. При объединении отдельных ЗУ в систему к этим факторам добавляется еще целый ряд, связанный со взаимодействием ЗУ между собой в составе системы памяти.

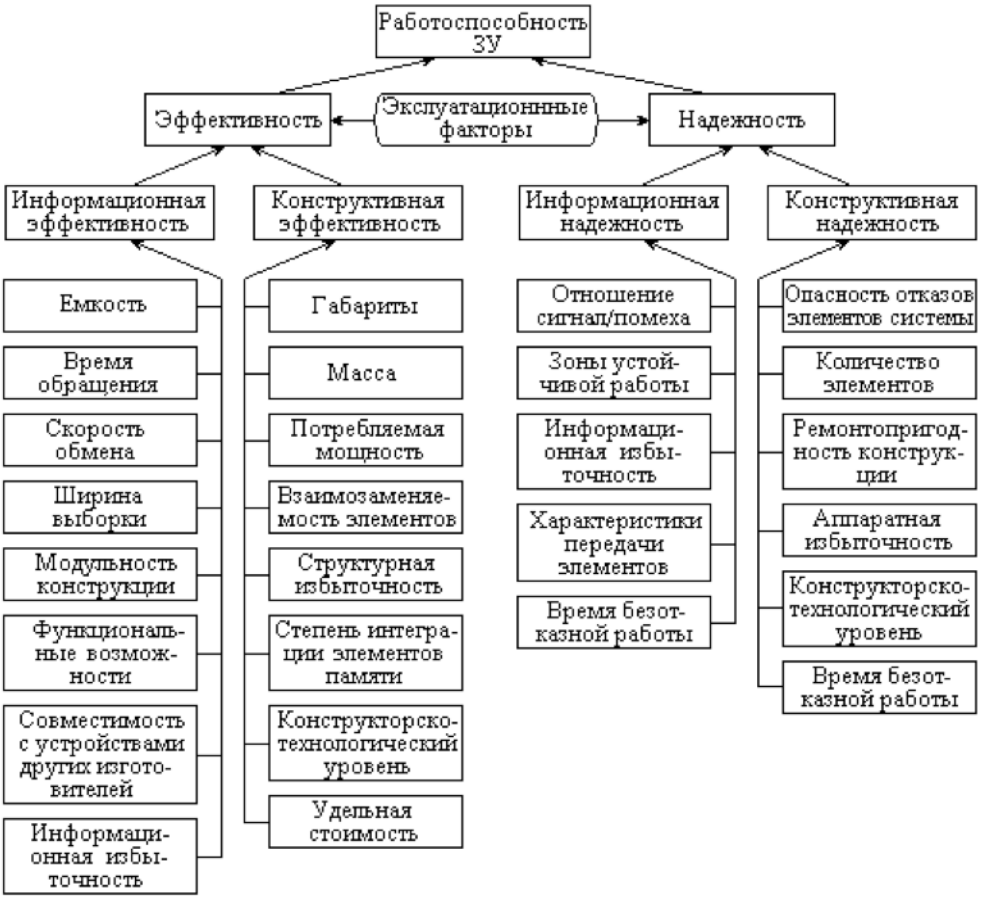


Рис. 5.2. Факторы, определяющие работоспособность ЗУ

С целью более полного учета характера функционирования и окружения СП при выборе критерия ее оценки следует рассматривать эту систему как компоненту вычислительной машины (системы), ориентируясь на назначение последней.

Любой критерий оценки должен включать основные характеристики оцениваемой системы, к которым относятся:

- емкость системы памяти,
- среднее время обращения к ней,
- пропускная способность,
- стоимость,
- надежность.

Ряд характеристик, например, радиационная устойчивость, габариты, масса, энергопотребление, в типовых применениях могут не учитываться. Хотя, если речь идет, например, о мобильных системах, последние три из названных характеристик имеют важное значение.

Емкость $E_{СП}$ системы памяти можно рассматривать в двух аспектах:

- либо как сумму объемов всех ЗУ, входящих в состав СП (технической емкости),
- либо как количество информации (программ и данных/эффективной емкости), которое можно разместить в системе.

В первом случае можно говорить о технической емкости СП, во втором – об эффективной емкости. Понятно, что эффективная емкость всегда меньше технической, так как она определяется не только собственно составом СП, но и методами организации хранения данных, методами управления памятью и др. Например, можно вспомнить о файловых системах, которые накладывают ограничения снизу на место на диске, занимаемое даже самым небольшим файлом.

Среднее время обращения $T_{обр}$ к СП можно определить через частоты обращений к отдельным устройствам системы и времена обращений $t_{обрi}$ к этим устройствам как

$$T_{обр} = \frac{\sum_i f_i t_{обр\,i}}{\sum_i f_i}$$

где f_i есть среднее количество обращений к i-му ЗУ в единицу времени. Очевидно, что в этом случае $T_{обр}$ в значительной степени зависит от относительных частот обращения к различным ЗУ, а не только от времени обращения к ним.

Средняя пропускная способность *B* системы памяти – количество информации, которое можно передать в СП или извлечь из нее в единицу времени. В общем виде ***B*** можно определить, усреднив (взвешенно) пропускные способности отдельных ЗУ, входящих в состав СП.

Стоимость $C_{СП}$ системы памяти определяется как сумма стоимостей всех входящих в ее состав ЗУ, контроллеров и дополнительных аппаратных средств, используемых для управления памятью. Строго говоря, некоторые средства управления памятью, как и сами ЗУ, могут быть интегрированы в процессор или в системные микросхемы (микросхемы чипсета или др.). В этом случае собственно стоимость ЗУ и средств управления ими приходится определять путем сравнения с аналогичными микросхемами, обладающими другими параметрами, или приближенно.

Надежность СП определяется надежностью составляющих ее блоков.

Все характеристики систем памяти взаимосвязаны между собой и имеют противоречивый характер. Например, уменьшение времени обращения к СП связано с использованием более быстродействующих, а, следовательно, и дорогостоящих ЗУ. Увеличения пропускной способности дисковых ЗУ можно достичь, используя аппаратные или алгоритмические методы диспетчеризации, что приводит либо к росту стоимости и снижению надежности СП, либо к увеличению расходов времени на работу операционной системы.

5.3. Основные характеристики современных запоминающих устройств (ЗУ). Классификация ЗУ

Запоминающие устройства (ЗУ) характеризуются рядом параметров, определяющих возможные области применения различных типов таких устройств. К основным параметрам, по которым производится наиболее общая оценка ЗУ, относятся их

- **информационная емкость (Е),**
- **время обращения (Т)** и
- **стоимость (С).**

Под **информационной емкостью ЗУ** понимают количество информации, измеряемое в байтах, килобайтах, мегабайтах или гигабайтах, которое может храниться в запоминающем устройстве.

Обычно информационная емкость учитывает только полезный объем хранимой информации, который не включает объем памяти, расходуемый на служебную информацию, контрольные разряды или байты, резервные области (например, интервал между концом дорожки диска и ее началом), дорожки синхросигналов и пр.

Время обращения к ЗУ различных типов определяется по-разному.

В качестве примера можно рассмотреть оперативные ЗУ и жесткие диски.

Оперативные ЗУ обычно реализуются как ЗУ с произвольным доступом. Это означает, что доступ к данным, физически организованным в виде двумерного массива (матрицы элементов памяти), производится с помощью схем дешифрации, выбирающих нужные строку и столбец массива по их номерам (адресам), как показано на рис. 5.3. Поэтому время $T_{обр}$ обращения к ним определяется, в случае отсутствия дополнительных этапов (таких, например, как передача адреса за два такта), временем срабатывания схем дешифрации адреса и собственно временами записи или считывания данных.

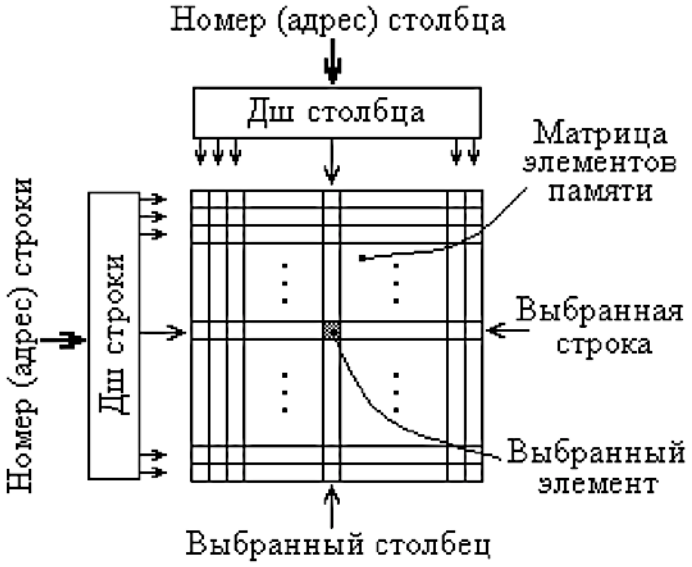


Рис. 5.3. Произвольный доступ к массиву элементов памяти ЗУ

Емкости оперативных ЗУ – порядка 256 Мб – 8 Гб.

Процесс обращения (чтения или записи) к жесткому диску представлен на рис.5.4. Он включает в себя 3 этапа:

1. перемещение блока головок чтения/записи на нужную дорожку (*а*),
2. ожидание подхода требуемого сектора под головки чтения/записи (*б*)
3. собственно, передача данных, считываемых/записываемых.

Каждый из этих этапов занимает определенное время, входящее в общее время обращения к диску. Все этапы так или иначе связаны с механическими перемещениями, поэтому их времена сравнительно велики и составляют величины порядка единиц миллисекунд.

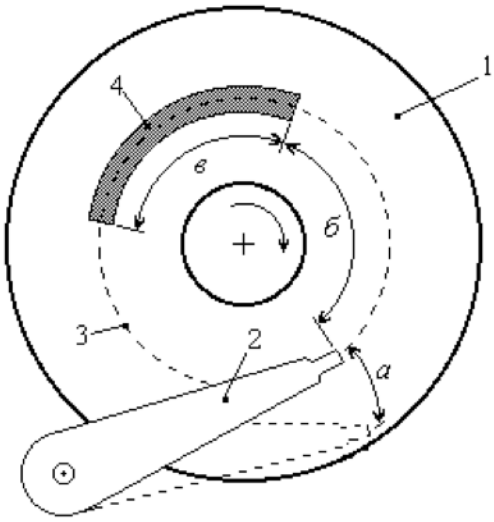


Рис. 5.4. Обращение к жесткому диску
(1 – пластина диска, 2 – блок чт/зп, 3 – дорожка (цилиндр), 4 – файл, *а* – поиск дорожки (перемещение блока головок чт/зп), *б* – ожидание подхода фа ла под блок головок, *в* – передача данных)

Время перемещения блока головок, обычно называемое изготовителями дисков временем поиска (seek time), зависит от количества дорожек, на которое надо переместить блок головок. Минимальное время затрачивается на перемещение блока головок на соседнюю дорожку (цилиндр) – составляет порядка 1–2 мс. Максимальное время требуется на перемещение блока головок от крайней дорожки к центральной или наоборот – может составлять порядка 15–20 мс. Среднее время поиска (перемещения головок) составляет порядка 8–10 мс.

Время ожидания подвода файла (точнее, его первого сектора) под блок головок производители называют также временем задержки (*latency time*). Это время в среднем равно времени половины оборота диска, что, например, при скорости вращения (шпинделя) диска 7200 оборотов/мин, или 120 оборотов/с, составляет 4,2 мс.

Наконец, время передачи данных зависит от количества передаваемых данных (размера файла, если он располагается целиком на последовательных секторах одной дорожки диска) и скорости передачи. Из-за зависимости этого времени от размера файла и его размещения на диске в качестве характеристики диска используют скорость передачи данных (*transfer rate*). Эта скорость определяется как параметрами тракта связи с ЭВМ, так и скоростью считывания

данных с диска или записи данных на диск. Обычно пользуются именно этими параметрами, так как каналы передачи достаточно быстрые, чтобы снижать скорость передачи, а диски имеют буферные ЗУ (кэш диска), скорость обмена данными с которым заметно превышает скорость считывания с диска или записи на диск.

В свою очередь, скорость обмена с диском определяется скоростью его вращения и плотностью записи информации на него. Обе эти величины непрерывно возрастают с развитием технологий изготовления жестких дисков.

Плотность записи информации на диск удваивалась примерно каждый год – полтора.

Стоимость запоминающих устройств также представляет собой важную характеристику. Именно она является одной из причин иерархической организации памяти ЭВМ.

Действительно, хорошо иметь быструю и емкую память. Нужно, чтобы она была и относительно дешевой. Понятно, что эти параметры противоречивы. Поэтому в ЭВМ и строят иерархию памяти.

Определения дорогие и дешевые понимаются не в абсолютном, а в относительном измерении, исходя из стоимости хранения единицы информации (удельной стоимости) в ЗУ. Стоимость хранения 1 Мбайта информации в оперативных ЗУ и на жестких дисках различаются примерно в 100 раз.

Конечно, помимо емкости, времени обращения и стоимости, существуют и другие характеристики памяти такие, как:

- надежность;
- энергопотребление;
- габариты;
- время хранения информации;
- способность сохранять ее при отключении питания и другие.

При определенных условиях эти характеристики могут иметь важное значение. Например, *для ноутбуков энергопотребление и габариты* играют существенную роль, что при обеспечении требуемых значений этих показателей приводит к более высокой стоимости устройств такого класса. Напротив, *для серверов* на первый план выдвигается *требование надежности* сохранения информации.

Перейдем к классификации запоминающих устройств. Существует большое количество различных типов ЗУ, используемых в ЭВМ и системах. Эти устройства различаются рядом признаков:

- принципом действия,
- логической организацией,
- конструктивной и
- технологической реализацией,
- функциональным назначением и т.д.

Большое количество существующих типов ЗУ обусловливает различия в структурной и логической организации (систем) памяти ЭВМ. ***Требуемые***

характеристики памяти достигаются ***не только за счет применения ЗУ с соответствующими характеристиками, но*** в значительной степени ***за счет особенностей ее структуры и алгоритмов функционирования***.

Память ЭВМ почти всегда является "узким местом", ограничивающим производительность компьютера. Поэтому в ее организации используется ряд приемов, улучшающих временные характеристики памяти и, следовательно, повышающих производительность ЭВМ в целом.

Классификация запоминающих устройств и систем памяти позволяет выделить общие и характерные особенности их организации, систематизировать базовые принципы и методы, положенные в основу их реализации и использования.

Один из возможных вариантов классификации ЗУ представлен на рис. 5.5 В нем устройства памяти подразделяются по двум основным критериям: по ***функциональному назначению*** (роли или месту в иерархии памяти) и ***принципу организации.***

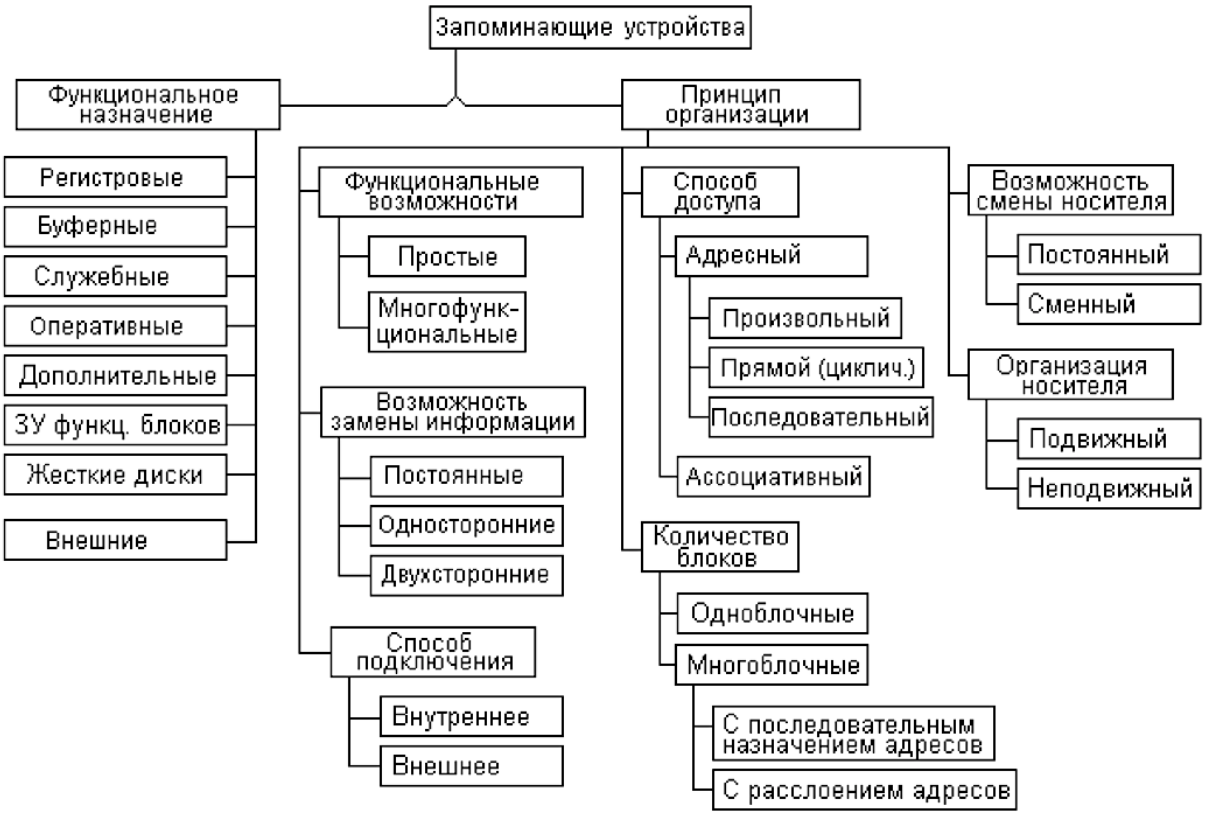


Рис. 5.5. Классификация запоминающих устройств

5.4. Полупроводниковые запоминающие устройства: организация ЗУ с произвольным доступом

Полупроводниковые ЗУ в настоящее время представляют собой большой класс запоминающих устройств, различных по своим функциональным и техническим характеристикам, широко используемых в качестве внутренних ЗУ ЭВМ. Но этим их использование не ограничивается. Подавляющее большинство

электронной и бытовой техники переходит на цифровые методы представления данных (не только текстовых, но и аудио, графических и видео) и управления (использование микроконтроллеров). Различные сферы применения накладывают свои особенности на реализацию полупроводниковых ЗУ, однако это чаще касается их конструктивных особенностей, а принципы построения одинаковы.

Высокое быстродействие полупроводниковых ЗУ обусловливает то, что большинство из них имеет организацию с произвольным доступом. Это же высокое быстродействие определяет и основные области применения полупроводниковых ЗУ в ЭВМ: кэш-память и оперативная память.

В ***статических ЗУ*** (**Static Random Access Memory – SRAM**) в качестве элемента памяти используется триггер, поэтому статические ЗУ обладают меньшей плотностью хранения информации: емкость типовых микросхем статических ЗУ начала 2000-х годов не превосходила 16 Мбит.

Однако триггер со времен первых компьютеров был и остается самым быстродействующим элементом памяти. Поэтому статическая память позволяет достичь наибольшего быстродействия, обеспечивая время доступа в единицы и даже десятые доли наносекунд, что и обусловливает ее использование в ЭВМ, главным образом, в высших ступенях памяти – кэш-памяти всех уровней.

Главными недостатками статической памяти являются ее относительно высокие стоимость и энергопотребление.

Конечно, в зависимости от используемой технологии, память будет обладать различным сочетанием параметров быстродействия и потребляемой мощности. Например, статическая память, изготовленная по КМОП-технологии (КМОП – комплементарный металло-оксидный полупроводник) может быть светочувствительной.

Основными разновидностями статической памяти SRAM с точки зрения организации ее функционирования являются:

- ***асинхронная*** (Asynchronous) SRAM,
- ***синхронная пакетная*** (Synchronous Burst) SRAM и
- ***синхронная конвейерно-пакетная*** (Pipeline Burst) SRAM память.

Синхронная пакетная статическая память (**SBSRAM**) ориентирована на выполнение пакетного обмена информацией, который характерен для кэш-памяти. Эта память включает в себя внутренний счетчик адреса, предназначенный для перебора адресов пакета, и использует сигналы синхронизации, как и синхронная DRAM память.

Следующим шагом в развитии статической памяти явилась ***конвейерно-пакетная память*** PBSRAM, обеспечивающая более высокое быстродействие, чем SBSRAM. В нее были введены дополнительные внутренние буферные регистры данных (здесь можно провести аналогию с EDO DRAM памятью) адреса, а в ряде модификаций предусмотрена возможность передачи данных на двойной скорости по переднему и заднему фронтам синхросигнала и используются сдвоенные внутренние тракты записи и чтения. Это позволило получить время обращения

порядка 2-3 нс и обеспечить передачу данных пакета без задержек на частотах шины более 400 Мгц.

Внутренняя логика позволяет переключаться с циклов чтения на циклы записи и наоборот без дополнительных задержек, кроме того, анализируется совпадение адресов записи и чтения для исключения избыточных операций.

Как уже отмечалось, в качестве оперативных ЗУ в настоящее время чаще используются ***динамические ЗУ с произвольным доступом*** (**DRAM**). Такое положение обусловлено тем, что недостатки, связанные с необходимостью регенерации информации в таких ЗУ и относительно невысоким их быстродействием, с лихвой компенсируются другими показателями: малыми размерами элементов памяти и, следовательно, большим объемом микросхем этих ЗУ, а также низкой их стоимостью.

Широкое распространение ЗУ этого типа проявилось также и в разработке многих его разновидностей: асинхронной, синхронной, RAMBUS и других. Перечислим основные разновидности:

1. Синхронная динамическая память DDR SDRAM

   обеспечивающая двойную скорость передачи данных (*DDR – Double* или *Dual Data Rate*), в которой за один такт осуществляются две передачи данных – по переднему и заднему фронтам каждого синхроимпульса. Во всем остальном эта память работает аналогично обычной SDRAM памяти. Времена задержек *CAS Latency* для DDR SDRAM могут быть 2 и 2,5 такта

2. Синхронная динамическая память SDRAM

   иногда называют SDR SDRAM – *Single Data Rate.*

3. Асинхронная динамическая память DRAM
4. Динамическая память RDRAM

Независимо от того, какова конкретная модификация динамической памяти, запоминающие конденсаторы ее запоминающих элементов разряжаются из-за наличия токов утечки. Постоянная разрядка, как известно, зависит от емкости запоминающего конденсатора и сопротивления цепи тока утечки и может различаться для разных модификаций. Время, в течение которого информация сохраняется в элементе памяти, составляет до нескольких десятков миллисекунд.

Это приводит к необходимости периодического (с периодом не больше, чем время сохранения информации) восстановления зарядов емкостей. Такая процедура и получила название регенерации (***refresh***) динамической памяти. Выполняется она одновременно для целой строки матрицы (банка) элементов памяти, поскольку регенерировать информацию по элементам или по словам (по 8 байт) слишком долго.

Модули динамической полупроводниковой памяти прошли эволюцию от набора микросхем, устанавливаемых на системной плате и заметных по своему регулярному расположению (несколько смежных рядов одинаковых микросхем) до отдельных небольших плат, вставляемых в стандартный разъем (слот) системной платы. Первенство в создании таких модулей памяти обычно относят к фирме IBM.

Основными разновидностями модулей динамических оперативных ЗУ с момента их оформления в виде самостоятельных единиц были:

- 30-контактные однобайтные модули SIMM (DRAM)
- 72-контактные четырехбайтные модули SIMM (DRAM)
- 168-контактные восьмибайтные модули DIMM (SDRAM)
- 184-контактные восьмибайтные модули DIMM (DDR SDRAM)
- 184-контактные (20 из них не заняты) двухбайтные модули RIMM RDDRAM).

Сокращение SIMM означает Single In-Line Memory Module – модуль памяти с одним рядом контактов, так как контакты краевого разъема модуля, расположенные в одинаковых позициях с двух сторон платы, электрически соединены. Соответственно DIMM значит Dual In-Line Memory Module – модуль памяти с двумя рядами контактов. А вот RIMM означает Rambus Memory Module – модуль памяти типа Rambus.

5.5. Постоянные запоминающие устройства: разновидности

Постоянные запоминающие устройства (**ПЗУ** или **Read Only Memory - ROM**), которые также часто называют энергонезависимыми (или Non Volatile Storage), обеспечивают сохранение записанной в них информации и при отсутствии напряжения питания. Конечно, под такое определение подпадают и память на жестких и гибких дисках, и компакт диски, и некоторые другие виды ЗУ.

Однако, говоря о постоянных ЗУ, обычно подразумевают устройства памяти с произвольным адресным доступом. Такие ЗУ могут строиться на различных физических принципах и обладать различными характеристиками не только по емкости и времени обращения к ним, но и по возможности замены записанной в них информации.

К началу 2000-х годов наибольшее распространение получили полупроводниковые ПЗУ, элементы памяти которых используют различные модификации диодов и транзисторов и изготавливаются по интегральной технологии.

Непосредственными предшественниками таких ЗУ были магнитные (трансформаторные) ПЗУ, информация в которые записывалась соответствующей прокладкой (прошивкой) проводников ферритовых сердечников. Это обеспечивало при требовавшихся в то время емкостях высокую надежность этих ЗУ в самых тяжелых (в электромагнитном отношении) условиях.

Известны также емкостные и индуктивные ПЗУ, в которых использовались проводники специальной формы, образующие емкостные или индуктивные связи.

В настоящее время исследуются и другие принципы реализации постоянных ЗУ, в некотором смысле возвращающиеся к магнитным и конденсаторным схемам, но на другом уровне развития технологий.

Запись информации в ПЗУ, как правило, существенно отличается от считывания по способу и времени выполнения. Процесс записи для полупроводниковых постоянных ЗУ получил также название “прожига” или

программирования, первое из которых связано со способом записи, сводящимся к разрушению (расплавлению, прожигу) соединительных перемычек в чистом ЗУ.

В полупроводниковых ПЗУ в качестве элементов памяти, точнее, в качестве нелинейных коммутирующих и усилительных элементов обычно используются транзисторы. Они объединены в матрицу, выборка данных из которой производится по строкам и столбцам, соответствующим указанному адресу, так же, как и в других ЗУ с произвольным доступом. Один из возможных вариантов структурной схемы полупроводникового ПЗУ, представлен на рис. 5.6. Строго говоря, непосредственно запоминание информации в этом ПЗУ осуществляется плавкой перемычкой, а транзисторы выполняют роль ключей-усилителей. Плавкая перемычка может быть изготовлена из нихрома, поликристаллического кремния или других материалов. В зависимости от того, как именно работает усилитель считывания (в режиме повторителя или инвертора), наличие перемычки соответствует записи “1” или “0”. Разрушение перемычки (импульсом сильного тока) приводит к записи значения, обратного исходному.

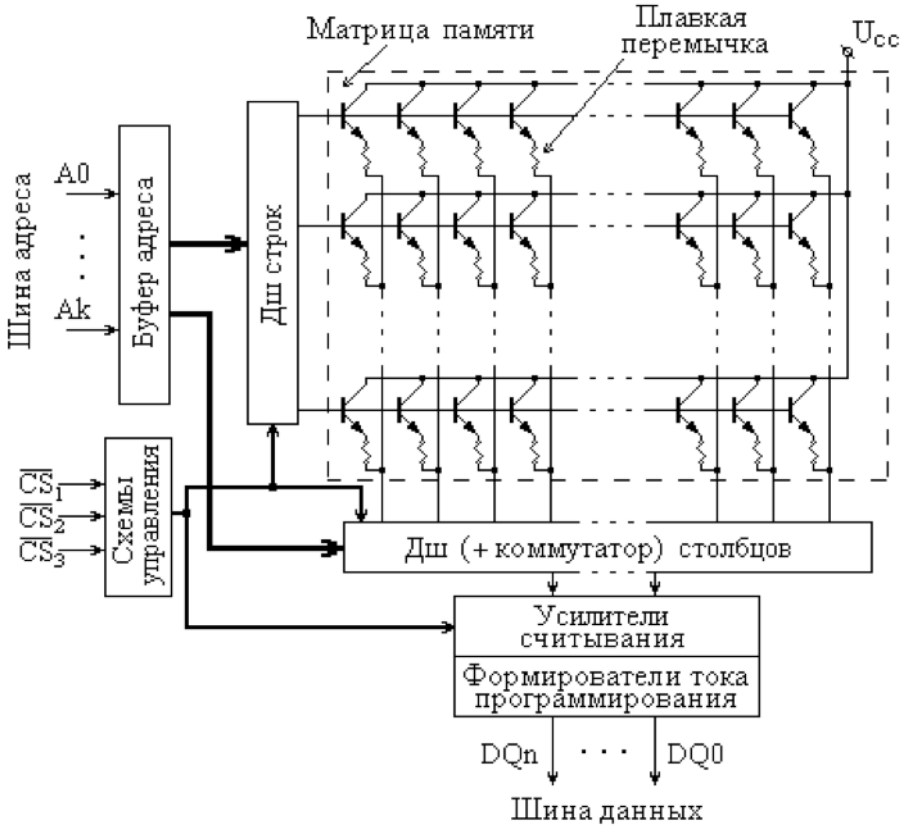


Рис. 5.6. Вариант структурной схемы ПЗУ с однократным программированием

Различают две большие группы ПЗУ: программируемые изготовителем и программируемые пользователем.

ЗУ первой группы, называемые иначе масочными, обычно выпускаются большими партиями. Информация в них заносится в процессе изготовления этих ЗУ на заводах: с помощью специальной маски в конце технологического цикла на кристалле формируется соответствующая конфигурация соединений. Такие ЗУ оказываются наиболее дешевыми при массовом изготовлении. Их обычно

используют для хранения различных постоянных программ и подпрограмм, кодов, физических констант, постоянных коэффициентов и пр.

В ПЗУ, программируемые пользователем, информация записывается после их изготовления самими пользователями. При этом существуют два основных типа таких ЗУ: однократно программируемые и перепрограммируемые.

Нетрудно вспомнить, что аналогичные разновидности имеются и у CDROM, которые, по существу, являются ПЗУ (ROM), изготавливаемыми на основе другого физического принципа.

Наиболее простыми являются однократно программируемые ПЗУ. В этих ЗУ запись как раз и производится посредством разрушения соединительных перемычек между выводами транзисторов и шинами матрицы (хотя есть и несколько иные технологии). Изображение программируемого ПЗУ на функциональной схеме показано на рис. 5.7.

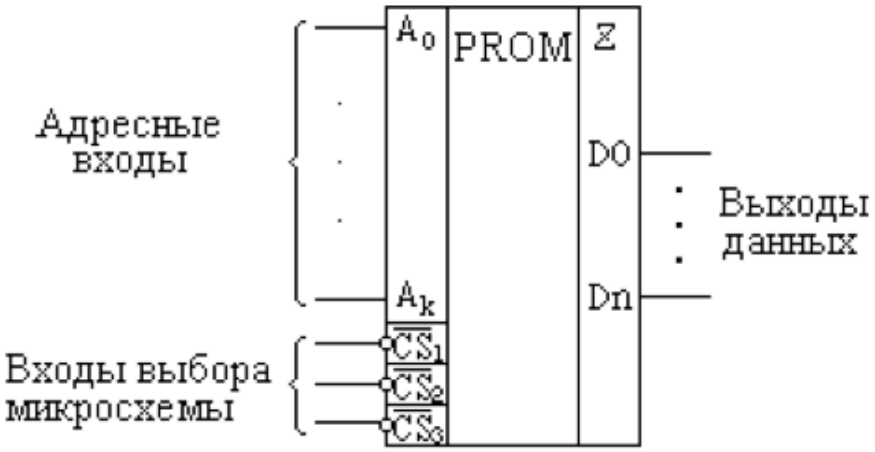


Рис. 5.7. Условное изображение программируемого ПЗУ на функциональных схемах

Перепрограммируемые ПЗУ позволяют производить в них запись информации многократно. Конечно, в таких ЗУ должен использоваться иной принцип, чем разрушение перемычек в процессе записи. Распространенные технологические варианты используют МОП-транзисторы со сложным затвором (составным или “плавающим”), который способен накапливать заряд, снижающий пороговое напряжение отпирания транзистора, и сохранять этот заряд при выключенном питании. Программирование таких ПЗУ и состоит в создании зарядов на затворах тех транзисторов, где должны быть записаны данные (обычно “0”, так как в исходном состоянии в таких микросхемах записаны все “1”).

Перед повторной записью требуется произвести стирание ранее записанной информации. Оно производится либо электрически, подачей напряжения обратной полярности, либо с помощью ультрафиолетового света. У микросхем последнего типа имелось круглое окошечко из кварцевого стекла, через которое и освещался кристалл при стирании.

Параметры постоянных ЗУ соответствуют технологическим нормам своего времени. В начале 2000-х годов типовые емкости микросхем постоянной памяти с

масочным программированием составляли порядка 32-128 Мбит, а времена обращения превышали аналогичные показатели оперативной памяти и для различных модификаций достигали до 100 нс.

5.6. Регистровая память

Сверхоперативные ЗУ (в настоящее время это кэш-память) используются для хранения небольших объемов информации и имеют значительно меньшее время (в 2 - 10 раз) считывания/записи, чем основная память. СОЗУ (или кэш) обычно строятся на регистрах и регистровых структурах.

Регистр представляет собой электронное устройство, способное хранить занесенное в него число неограниченно долго (при **включенном питании**). Наибольшее распространение получили регистры на статических триггерах.

По назначению регистры делятся на *регистры хранения* и регистры сдвига. Информация в регистры может заноситься и считываться либо параллельно, *сразу* всеми разрядами, либо последовательно, через один из крайних разрядов с последующим сдвигом занесенной информации.

Сдвиг записанной в регистр информации может производиться вправо или влево. Если регистр допускает сдвиг информации в любом направлении, он называется ***реверсивным.***

Регистры могут быть объединены в единую структуру. Возможности такой структуры определяются способом доступа и адресации регистров.

Если к любому регистру можно обратиться для записи/чтения по его адресу, такая регистровая структура образует ***СОЗУ с произвольным доступом***.

Безадресные регистровые структуры могут образовывать два вида устройств памяти: магазинного типа и память с выборкой по содержанию (***ассоциативные ЗУ***).

Память магазинного типа образуется из последовательно соединенных регистров (см. рис. 5.8).

Если запись в регистровую структуру (рис. 5.8,а) производится через один регистр, а считывание — через другой, то такая память является аналогом магазинной памяти и работает по принципу “первым вошел - первым вышел” (FIFO — first input, first output).

Если же запись и чтение осуществляются через один и тот же регистр (рис. 5.8,б), такое устройство называется *стековой памятью,* работающей по принципу “первым вошел — последним вышел” (FILO — first input, last output). При записи числа в стековую память сначала содержимое стека сдвигается в сторону последнего, К-го регистра (если стек был полностью заполнен, то число из К-го регистра теряется), а затем число заносится в вершину стека — регистр 1. Чтение осуществляется тоже через вершину стека, после того как число из вершины прочитано, стек сдвигается в сторону регистра 1.

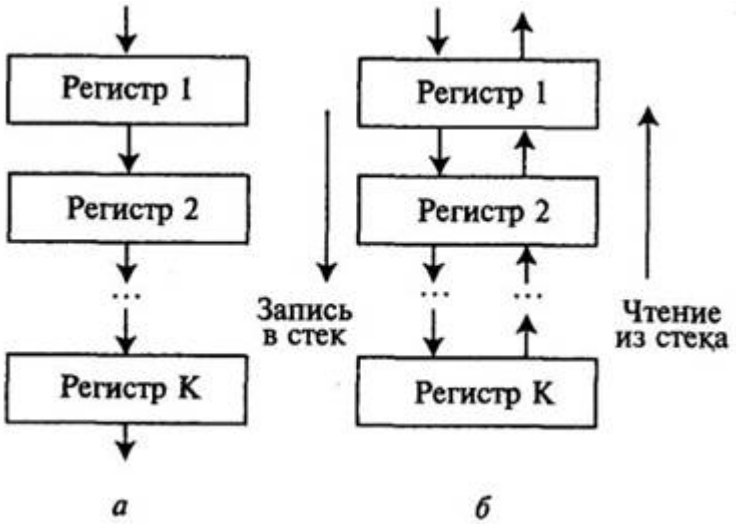


Рис. 5.8. Регистровая структура магазинного типа: *а* - типа FIFO; *б* - типа FILO

Стековая память получила широкое распространение. Для ее реализации в ЭВМ разработаны специальные микросхемы. Но часто работа стековой памяти ***эмулируется*** в основной памяти ЭВМ: с помощью программ операционной системы выделяется часть памяти под стек (в IBM PC для этой цели выделяется 64 Кбайта). Специальный регистр микропроцессора (указатель стека) постоянно хранит адрес ячейки ОП, выполняющей функции вершины стека. Чтение числа всегда производится из вершины стека, после чего указатель стека изменяется и указывает на очередную ячейку стековой памяти (т.е. фактически стек остается неподвижным, а перемещается вершина стека). При записи числа в стек сначала номер ячейки в указателе стека модифицируется так, чтобы он указывал на очередную свободную ячейку, после чего производится запись числа по этому адресу. Такая работа указателя стека позволяет реализовать принцип “первым вошел - последним вышел”. В стек может быть загружен в определенной последовательности ряд данных, которые впоследствии считываются из стека уже в обратном порядке, на этом свойстве построена система арифметических преобразований информации, известная под названием “логика Лукашевича”.

Память с выборкой по содержанию является безадресной. Обращение к ней осуществляется по ***специальной маске***, которая содержит поисковый образ. Информация считывается из памяти, если часть ее соответствует поисковому образу, зафиксированному в маске. Например, если в такую память записана информация, содержащая данные о месте жительства (включая город), и необходимо найти сведения о жителях определенного города, то название этого города помещается в маску и дается команда **чтение** - из памяти выбираются все записи, относящиеся к заданному городу.

В микропроцессорах ассоциативные ЗУ используются в составе кэш-памяти для хранения адресной части команд и операндов исполняемой программы. При этом нет необходимости обращаться к ОП за следующей командой или требуемым операндом: достаточно поместить в маску необходимый адрес, если искомая информация имеется в СОЗУ, то она будет сразу выдана. Обращение к ОП будет необходимо лишь при отсутствии требуемой информации в СОЗУ. За счет такого использования СОЗУ сокращается число обращений к ОП, а это позволяет

экономить время, так как обращение к СОЗУ требует в 2 - 10 раз меньше времени, чем обращение к ОП.

Уровни кеша

Кеш-память уровня *N*+1 всегда больше по размеру и медленнее по скорости обращения, чем кеш-память уровня *N*.

Самой быстрой памятью является кеш-память первого уровня **L1**-cache: **L1I**, **L1D - неотъемлемая часть процессора**, расположена на том же кристалле и входит в состав функциональных блоков. Без нее процессор не сможет функционировать. Память **L1** работает на частоте процессора и в общем случае обращение к ней может производиться каждый такт (зачастую возможно выполнять даже несколько чтений/записей одновременно), латентность доступа обычно равна **2-4 такта ядра**, объем этой памяти обычно невелик — **не более 64Кб**.

Второй по быстродействию является **L2.** В отличие от L1 ее можно отключить с сохранением работоспособности процессора. Кеш второго уровня обычно расположена либо на кристалле, как и **L1,** либо в непосредственной близости от ядра, например, в процессорном картридже (только в слотовых процессорах), в старых процессорах ее располагали на системной плате. Объем **L2** больше — **от 128Кб до 1—4Мб**. Обычно **латентность L2,** расположенной на кристалле ядра, составляет от **8 до 20 тактов ядра.**

Кеш третьего уровня **L3** наименее быстродействующий и обычно расположен отдельно от ядра ЦП, но он может быть очень внушительного размера и всё равно значительно быстрее, чем оперативная память.

Сегодня существует и **L4** уровень кэша, обыкновенно он расположен в отдельной микросхеме. Применение кэша 4 уровня оправдано только для высокопроизводительных серверов и мейнфреймов.

Объемы кеш-памяти: **L1до 128 Кб,**
L2 от 128 Кб до 1-12 Мб,
L3 свыше 24 Мб.

Проблема синхронизации между различными уровнями кэш-памяти (как одного, так и множества процессоров) решается когерентностью кэша.

Существует три варианта обмена информацией между кэш-памятью различных уровней, или, как говорят, кэш-архитектуры:

- **инклюзивная** – предполагает дублирование информации кэша верхнего уровня в нижнем (предпочитает фирма **Intel)**;
- **эксклюзивная** - предполагает уникальность информации, находящейся в различных уровнях кэша (предпочитает фирма **AMD)**;
- **неэксклюзивная** - уровни кэша связываются между собой произвольным образом.

На рис. 5.9 представлена схема расположения регистровой памяти, кеш-памяти и основной памяти, в которой обозначены место, объем и время доступа к каждой.

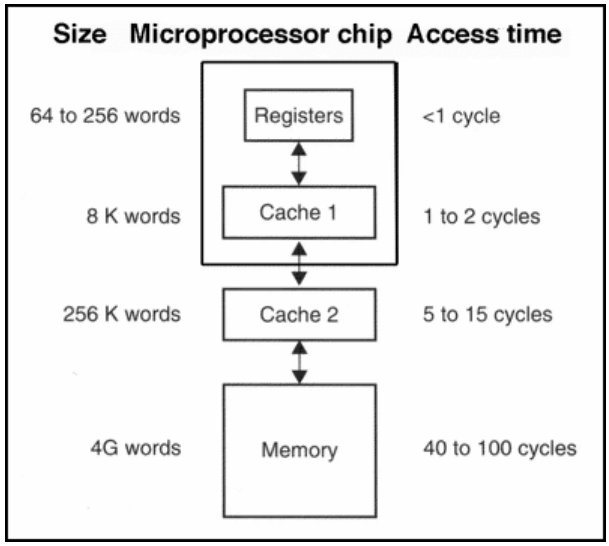
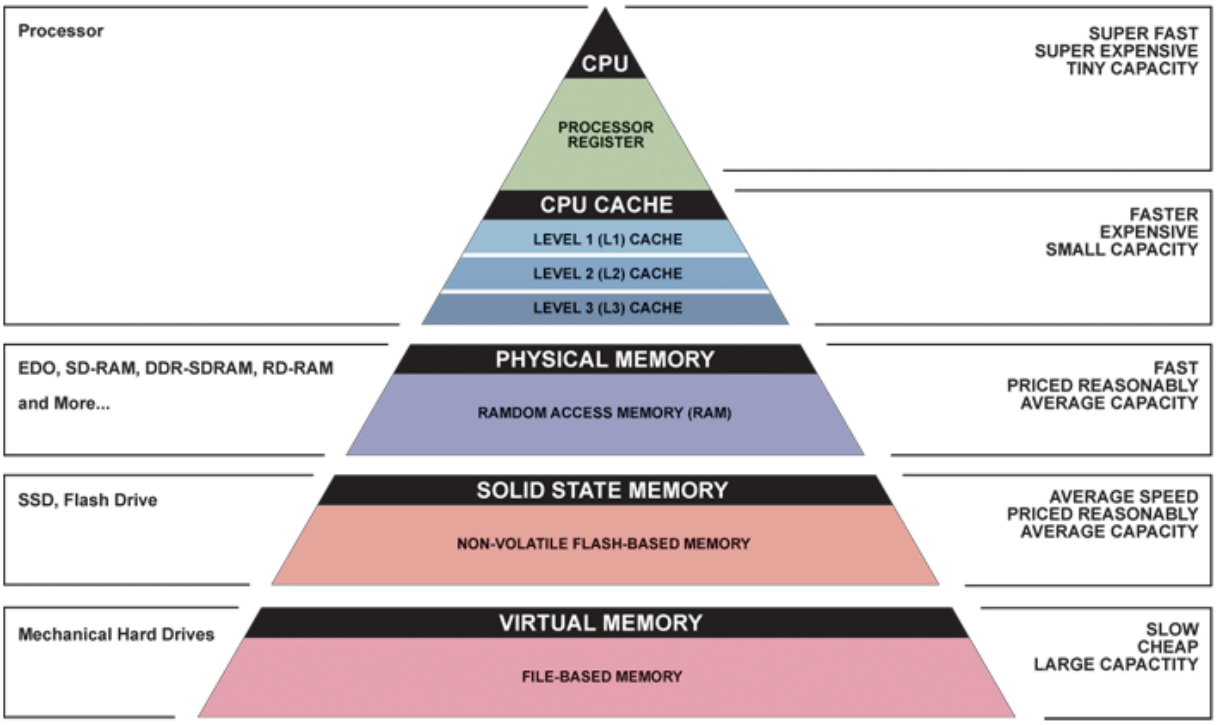


Рис. 5.9. Схема уровней памяти

На рис. 5.10 приведена полная иерархическая структура памяти компьютера с указанием места расположения, типа памяти, ее разновидностей и таких общих характеристик как скорости, стоимости и объема.



▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Рис. 5.10. Иерархическая структура компьютерной памяти

P.S. См. презентации по теме из папки «Дополнительные материалы»

Лекция 6. Устройства и принципы управления ЭВМ

Основные вопросы:

Любая система обработки данных всегда рассматривается как совокупность трех устройств:

- памяти;
- устройства обработки;
- устройства управления.

Устройство управления (УУ) предназначено для выработки управляющих сигналов, необходимых для выполнения любого действия, происходящего в системе обработки данных.

Классификация УУ:

1. По структурной организации:
   - Централизованные
   - Смешанные – централизованные + местные
   - Иерархические (в ВС)
2. По технической организации:
   - С жесткой логикой работы
   - Устройства микропрограммного управления

6.1. Устройства управления с жесткой логикой работы

Выполнение операций в машине сводится к элементарным преобразованиям информации (передача информации между узлами в блоках, сдвиг информации в узлах, логические поразрядные операции, проверка условий и т.д.) в логических элементах, узлах и блоках под воздействием функциональных управляющих сигналов блоков (устройств) управления. Элементарные преобразования, неразложимые на более простые, выполняются в течение одного такта сигналов синхронизации и называются **микрооперациями (МИО)**. Таким образом, микрооперация – элементарное действие, выполняемое тем или иным функциональным узлом ЭВМ. Набор микроопераций образует **микропрограмму.**

Известны два подхода к построению логики формирования функциональных импульсов. Один из них – аппаратный.

В аппаратных (схемных) устройствах управления каждой операции соответствует свой набор логических схем, выполненных на диодах, транзисторах и т.д. и определяющих, какой функциональный импульс (ФИ) и в каком такте должен быть возбужден. Т.е. **логические схемы вырабатывают определенные функциональные сигналы для выполнения микроопераций в определенные моменты времени**. При этом способе построения УУ реализация микроопераций

достигается за счет однажды соединенных между собой логических схем, поэтому ЭВМ с аппаратным устройством управления называют ЭВМ с жесткой логикой управления. Это понятие относится к фиксации системы команд в структуре связей ЭВМ и означает практическую невозможность каких-либо изменений в системе команд ЭВМ после ее изготовления.

Пусть некоторый ФИ должен появиться в такте *j* операции m при условии наличия переполнения сумматора или в такте *i* операции *n*. Требуемое действие будет выполнено, если подать сигналы, соответствующие указанным кодам операции, тактам и условиям на входы схем **И**, а выходы последних через схему **ИЛИ** соединить с формирователем ФИ (рис. 6.1).

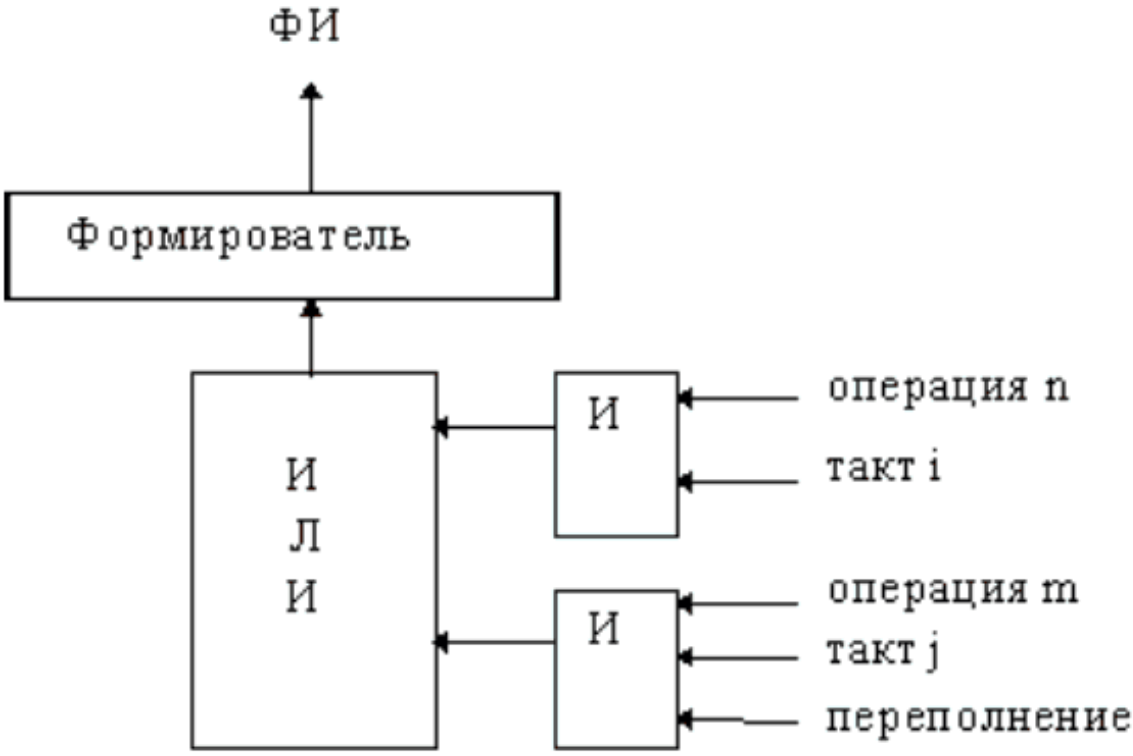


Рис. 6.1. Формирование функционального импульса

Такой принцип управления операциями получил название **"жесткой" или "запаянной" логики** и широко применяется во многих компьютерах.

Существует и другой принцип организации управления: каждой **микрооперации** ставится в соответствие слово (или часть слова), называемое **микрокомандой**[1] (**МИК**)и хранимое в памяти подобно тому, как хранятся в памяти команды обычного компьютера. Здесь команде соответствует микропрограмма, т.е. набор микрокоманд, указывающих, какие ФИ и в какой последовательности необходимо возбуждать для выполнения данной операции. Такой подход получил название **микропрограммирования или "хранимой логики".** Это подчеркивает тот факт, что в микропрограммном компьютере логика

[1] Микрокоманда – совокупность совместимых микроприказов, инициирующих выполнение микрооперации

управления реализуется не в виде электронной схемы, а в виде закодированной информации, находящейся в каком-то регистре.

6.2. Микропрограммное управление

Идея микропрограммирования, высказанная в 1951 г. Уилксом, нашла широкое применение в машинах серии IBM 360, когда появились надежные и быстродействующие ЗУ для хранения микропрограммы. За счет микропрограммного управления появилась возможность эмуляции системы команд старых моделей ЭВМ.

Достаточно долго неправильно понимались задачи и выгоды микропрограммирования.

Ценность микропрограммирования считалась в том, что каждый потребитель может сконструировать себе из МИК нужный ему набор операций в конкретной задаче. Замена наборов команд достигалась бы заменой информации в ЗУ без каких-либо переделок в аппаратуре. Однако в этом случае программисту необходимо было бы знать все тонкости работы инженера-разработчика компьютера. А основная тенденция развития ЭВМ в связи с автоматизацией программирования состоит в том, чтобы освободить программиста от детального изучения устройств компьютера и в максимальной степени приблизить язык компьютера к языку человека. Поэтому микропрограммные компьютеры считали трудными для пользователя.

Микропрограммный принцип приобрел актуальность, когда были:

- созданы односторонние (читающие) быстродействующие ЗУ с малым циклом памяти;
- микропрограммирование стало рассматриваться не как средство повышения гибкости программирования, а как метод построения системы управления процессором, удобный для инженера-разработчика компьютера.

Программист в своей работе может и не подозревать о микропрограммной структуре компьютера и использовать все средства программного обеспечения и языки программирования самого высокого уровня. Использование микропрограммного принципа позволяет облегчить разработку и изменение логики процессора.

С появлением программного доступа к состоянию процессора после выполнения каждой МИК обеспечивается возможность создания экономичной системы автоматической диагностики неисправностей и появляется способность к эмуляции, т. е. к выполнению на данной ЭВМ программы, составленной в кодах команд другого компьютера. Это достигается введением дополнительного набора МИК, соответствующих командам эмулируемого компьютера.

Эти возможности способствуют распространению методов микропрограммирования при построении УУ в современных компьютерах.

Микропрограмма записывается в специализированную память – память микропрограмм или микрокоманд.

При микропрограммной реализации УУ в его состав вводится ЗУ, каждый разряд выходного кода которого определяет появление определенного функционального сигнала управления. Поэтому каждой микрооперации ставится в соответствие свой информационный код – микрокоманда. Набор микрокоманд и последовательность их реализации обеспечивают выполнение любой сложной операции. Набор микроопераций называют микропрограммами. Способ управления операциями путем последовательного считывания и интерпретации микрокоманд из ЗУ (наиболее часто в виде микропрограммного ЗУ используют быстродействующие программируемые логические матрицы), а также использования кодов микрокоманд для генерации функциональных управляющих сигналов называют микропрограммным. ЭВМ с таким способом управления обычно называют микропрограммными или с хранимой (гибкой) логикой управления.

К микропрограммам предъявляют **требования функциональной полноты** и **минимальности. Первое требование** необходимо для обеспечения возможности разработки микропрограмм любых машинных операций, а **второе** связано с желанием уменьшить объем используемого оборудования. Учет фактора быстродействия ведет к расширению микропрограмм, поскольку усложнение последних позволяет сократить время выполнения команд программы.

Преобразование информации выполняется в универсальном арифметико-логическом блоке микропроцессора. Он обычно строится на основе комбинационных логических схем.

Для ускорения выполнения определенных операций вводятся дополнительно специальные операционные узлы (например, циклические сдвигатели). Кроме того, в состав микропроцессорного комплекта, построенного на БИС, вводятся специализированные оперативные блоки арифметических расширителей.

Операционные возможности микропроцессора можно расширить за счет увеличения числа регистров. Если в регистровом буфере закрепление функций регистров отсутствует, то их можно использовать как для хранения данных, так и для хранения адресов. Подобные регистры микропроцессора называются регистрами общего назначения (РОН). По мере развития технологии реально осуществлено изготовление в микропроцессоре 16, 32, 64 и более регистров.

В целом, принцип микропрограммного управления включает следующие позиции:

1. любая операция, реализуемая устройством, является последовательностью элементарных действий - микроопераций;
2. для управления порядком следования микроопераций используются логические условия;
3. процесс выполнения операций в устройстве описывается в форме алгоритма, представляемого в терминах микроопераций и логических условий, называемого микропрограммой;
4. микропрограмма используется как форма представления функции устройства, на основе которой определяются структура и порядок функционирования устройства во времени.

Принцип микропрограммного управления обеспечивает гибкость микропроцессорной системы и позволяет осуществлять проблемную ориентацию микро- и миниЭВМ.

Существуют два вида микропрограммного управления:

- горизонтальное микропрограммирование (схема с минимальным кодированием, когда требуется достичь максимальной скорости работы процессора);
- вертикальное микропрограммирование (используются сильно закодированные команды, но снижены аппаратные затраты на обработку микрокоманд).

6.2.1. Горизонтальное микропрограммирование

При горизонтальном микропрограммировании – каждому разряду МИК соответствует определенная микрооперация, выполняемая независимо от содержания других разрядов. Микропрограмма может быть представлена в виде матрицы n ×m, где n – число ФИ, m – количество МИК, т. е. строка соответствует одной МИК, а столбец – одной МИО (рис. 6.2).

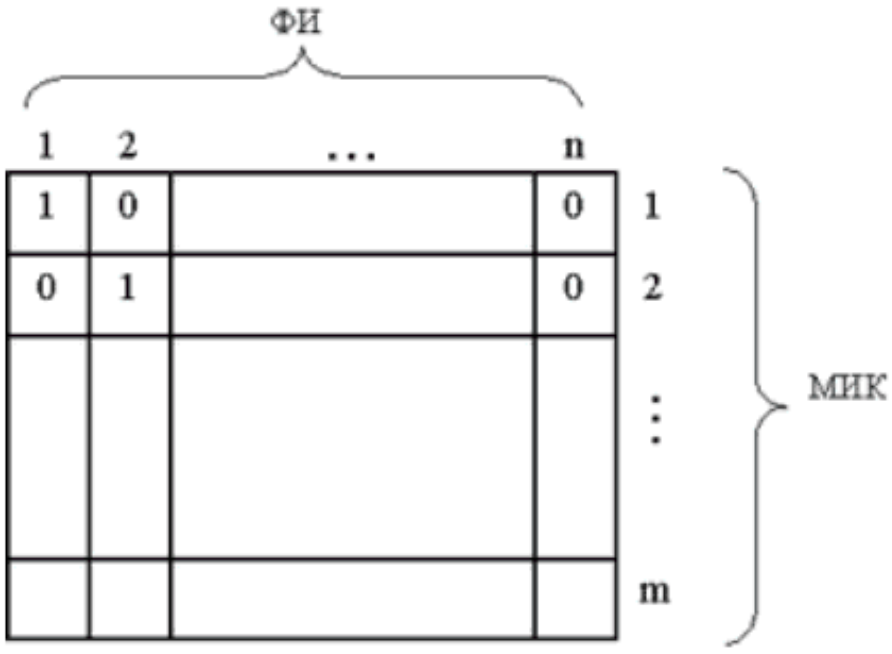


Рис. 6.2. Микропрограмма при горизонтальном микропрограммировании

Примерные значения разрядов МИК приведены на рис. 6.3.

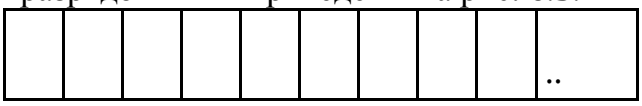


Рис. 6.3. Значение разрядов МИК (МИО):

1 – гашение сумматора; **2** – гашение указателя переполнения; **3** – обратный код сумматора; **4** – гашение регистра множителя частного; **5** – инвертирование знака; **6** – сдвиг содержимого сумматора влево; **7** – сдвиг содержимого сумматора вправо; **8** – увеличение содержимого сумматора на 1; **9** – чтение из ЗУ в сумматор.

Наличие "1" в пересечении какой-либо строки и столбца означает посылку ФИ в данную МИК, а наличие "0" – его отсутствие.

Размещение "1" в нескольких разрядах МИК означает выполнение нескольких МИО одновременно. Конечно, возбуждаемые МИО должны быть совместимы.

Пусть, например, разряды 9-разрядной МИК принимают следующие значения: 001001101. Тогда, если заданные разряды соответствуют семантике, указанной на рис. 6.3, то МИО, определяемые разрядами 9, 7 и 6, несовместимы.

Для расширения возможностей МИК иногда используют многотактный принцип исполнения МИК. При этом каждому разряду присваивается номер такта, в котором выполняется соответствующая ему МИО, т. е. здесь все совместимые МИО имеют один номер такта. Все остальные такты нумеруются в порядке их естественного выполнения. Однако универсальную нумерацию МИО в МИК указать затруднительно.

Достоинства горизонтального микропрограммирования:

- возможность одновременного выполнения нескольких МИО;
- простота формирования ФИ (без схем дешифрации).

Недостатки:

- большая длина МИК, так как число ФИ в современных компьютерах достигает нескольких сот, и соответственно большой объем ЗУ для хранения МИК;
- из-за ограничений совместимости операций, а также из-за последовательного характера выполнения алгоритмов операций лишь небольшая часть разрядов МИК будет содержать "1". В основном матрица будет состоять из нулей. Неэффективное использование ЗУ привело к малому распространению горизонтального микропрограммирования.

6.2.2. Вертикальное микропрограммирование

При вертикальном микропрограммировании каждая МИО определяется не состоянием одного разряда, а двоичным кодом, содержащимся в определенном поле МИК. Микрокоманда несколько напоминает формат обычных команд.

Отличие состоит в том, что:

- выполняется более элементарное действие – МИО вместо операции;
- адресная часть (в большинстве случаев) определяет не ячейку памяти, а операционный регистр процессора.

Формат МИК при вертикальном микропрограммировании приведен на рис. 6.4.

Рис. 6.4. Формат вертикальной МИК

Поля P_1 и P_2 в адресной части МИК указывают двоичные номера операционных регистров, содержимое которых участвует в одной операции. Одно из полей является одновременно и адресом результата. Таким образом, реализация арифметической или логической МИО, указанной в данной МИК, может быть выражена формулой $(P_1) \otimes (P_2) \rightarrow P_1$, или $(P_2) \rightarrow P_1$, где $\otimes$ – символ МИО.

Для МИК обращения к памяти поле P_1 указывает регистр, куда принимается информация, а P_2 – регистр, содержимое которого является адресом обращения к ЗУ. Указанный формат МИК не единственный.

Каждая МИК выполняет следующие функции:

- указывает выполняемую МИО;
- указывает следующую МИО через задание "следующего адреса";
- задает продолжительность МИК;
- указывает дополнительные действия – контроль и т. д.

Обычно в слове МИК имеются четыре зоны, соответствующие указанным функциям. Вообще говоря, некоторые из зон могут указываться неявно, например, выбор очередной МИК может осуществляться из следующей ячейки, продолжительность МИК может быть определена одинаковой для всех МИК и т. д.

Первыми компьютерами с микропрограммным управлением среди отечественных ЭВМ были: МИР, НАИРИ, среди зарубежных – IBM/360, Spectra 70 (70-ые годы).

Вывод:

Существуют два подхода к реализации управляющего блока процессора

- Аппаратную реализации УУ эффективнее использовать, если на первом плане быстродействие компьютера
- Микропрограммное УУ используется, если на первый план выступает требование гибкости реализации команд

6.3. Принципы управления

Вычислительные машины, помимо процессоров и основной памяти, образующих ее ядро, содержат многочисленные периферийные устройства (ПУ): внешние запоминающие устройства (ВЗУ) и устройства ввода/вывода (УВВ).

Передача информации с периферийного устройства в ЭВМ называется операцией ввода, а передача из ЭВМ в ПУ - операцией вывода.

Производительность и эффективность ЭВМ определяются не только возможностями ее процессора и характеристиками ОП, но и составом ПУ, их техническими данными и способами организации их совместной работы с ЭВМ.

В общем случае для организации и проведения обмена данными между двумя устройствами требуются **специальные средства:**

- специальные управляющие сигналы и их последовательности;
- устройства сопряжения;
- линии связи;
- программы, реализующие обмен.

Весь этот **комплекс линий и шин, сигналов, электронных схем, алгоритмов и программ, предназначенный для осуществления обмена информацией, называется *интерфейсом*.**

В зависимости от типа соединяемых устройств различаются:

- **внутренний интерфейс ЭВМ** (например, интерфейс системной шины, НМД), предназначенный для сопряжения элементов внутри системного блока ПЭВМ;
- **интерфейс ввода-вывода** - для сопряжения различных устройств с системным блоком (клавиатурой, принтером, сканером, мышью, дисплеем и др.);
- **интерфейсы межмашинного обмена** (для обмена между разными машинами) предназначены для сопряжения различных ЭВМ (например, при образовании вычислительных сетей, многопроцессорных и многомашинных комплексов);
- **интерфейсы "человек – машина"** – для обмена информацией между человеком и ЭВМ.

Если интерфейс обеспечивает обмен одновременно всеми разрядами передаваемой информационной единицы (чаще всего байта или машинного слова), он называется параллельным интерфейсом.

Внутренний интерфейс ЭВМ всегда делается параллельным или последовательно-параллельным (если одновременно передается не вся информационная единица, а ее часть, содержащая несколько двоичных разрядов).

Интерфейсы межмашинного обмена обычно **последовательные**, в которых обмен информацией производится по одному биту последовательно.

Для **параллельного и последовательно-параллельного интерфейса** необходимо, чтобы участники общения были связаны многожильным интерфейсным кабелем (**количество жил не меньше числа одновременно передаваемых разрядов – битов**). В **последовательных интерфейсах** участники

общения связываются друг с другом одно-двухпроводной линией связи, световодом, коаксиальным кабелем, радиоканалом.

Для каждого интерфейса характерно наличие специального аппаратурного комплекса (рис. 6.5).

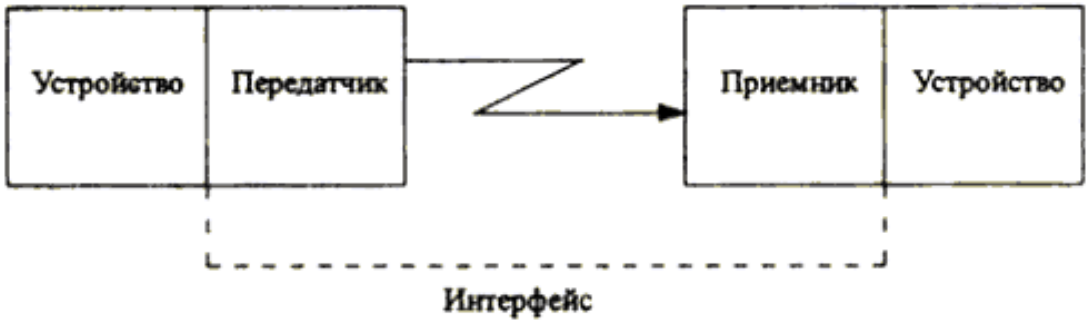


Рис. 6.5. Место интерфейса в аппаратурном комплексе

В зависимости от используемых при обмене программно-технических средств интерфейсы ввода-вывода делятся на два уровня: физический и логический (рис. 6.6).

Рис. 6.6. Логический и физический уровни интерфейсов ввода-вывода

В зависимости **от степени участия центрального процессора в обмене** данными в интерфейсах могут использоваться три способа управления обменом:

- режим сканирования (так называемый "**асинхронный" обмен**);
- **синхронный обмен**;
- **прямой доступ к памяти.**

Режим сканирования ("асинхронный" обмен)

Для внутреннего интерфейса ЭВМ режим сканирования предусматривает опрос центральным процессором ПУ: готово ли оно к обмену, если нет – продолжение опроса периферийного устройства (рис.6.7).

Операция пересылки данных логически слишком проста, чтобы эффективно загружать сложную быстродействующую аппаратуру процессора, в результате чего в режиме сканирования снижается производительность вычислительной машины.

Вместе с тем при пересылке блока данных процессору приходится для каждой единицы передаваемых данных (байт, слово) выполнять довольно много команд, чтобы обеспечить буферизацию данных, преобразование форматов, подсчет количества переданных данных, формирование адресов в памяти и т.п. В результате скорость передачи данных при пересылке блока данных даже через высокопроизводительный процессор может оказаться неприемлемой для систем управления, работающих в реальном масштабе времени.

Рис. 6.7. Алгоритм сканирования

Режим сканирования упрощает подготовку к обмену, но имеет рад недостатков:

- процессор постоянно задействован и не может выполнять другую работу;
- при большом быстродействии периферийного устройства процессор не успевает организовать обмен данными.

Синхронный режим

В синхронном режиме центральный процессор выполняет основную роль по организации обмена, но в отличие от режима сканирования не ждет готовности устройства, а осуществляет другую работу. Когда в нем возникает нужда, внешнее устройство с помощью соответствующего прерывания обращает на себя внимание центрального процессора.

Для быстрого ввода-вывода блоков данных и разгрузки процессора от управления операциями ввода-вывода используют прямой доступ к памяти (DMA – Direct Memory Access).

6.4. Прямой доступ к памяти

В режиме прямого доступа к памяти используется специализированное устройство – **контроллер прямого доступа к памяти (КПДП)**, который перед началом обмена программируется с помощью центрального процессора: в него передаются адреса основной памяти и количество передаваемых данных. Затем центральный процессор от контроллера прямого доступа к памяти отключается, разрешив ему работать, и до окончания обмена может выполнять другую работу. Об окончании обмена КПДП сообщает процессору. В этом случае участие центрального процессора косвенное. Обмен ведет контроллер прямого доступа к памяти.

На рис. 6.8 приведена схема взаимодействия устройств микропроцессорной системы в режиме ПДП.

Рис. 6.8. Взаимодействие устройств в режиме прямого доступа к памяти

Прямой доступ к памяти (ПДП):

- освобождает процессор от управления операциями ввода-вывода;
- позволяет осуществлять параллельно во времени выполнение процессором программы с обменом данными между внешним устройством и основной памятью;
- производит обмен данными со скоростью, ограничиваемой только пропускной способностью основной памяти и внешним устройством.

ПДП разгружает процессор от обслуживания операций ввода-вывода, способствует увеличению общей производительности ЭВМ, дает возможность ЭВМ работать в системах реального времени.

На рис. 6.9 представлена структурная схема контроллера ПДП.

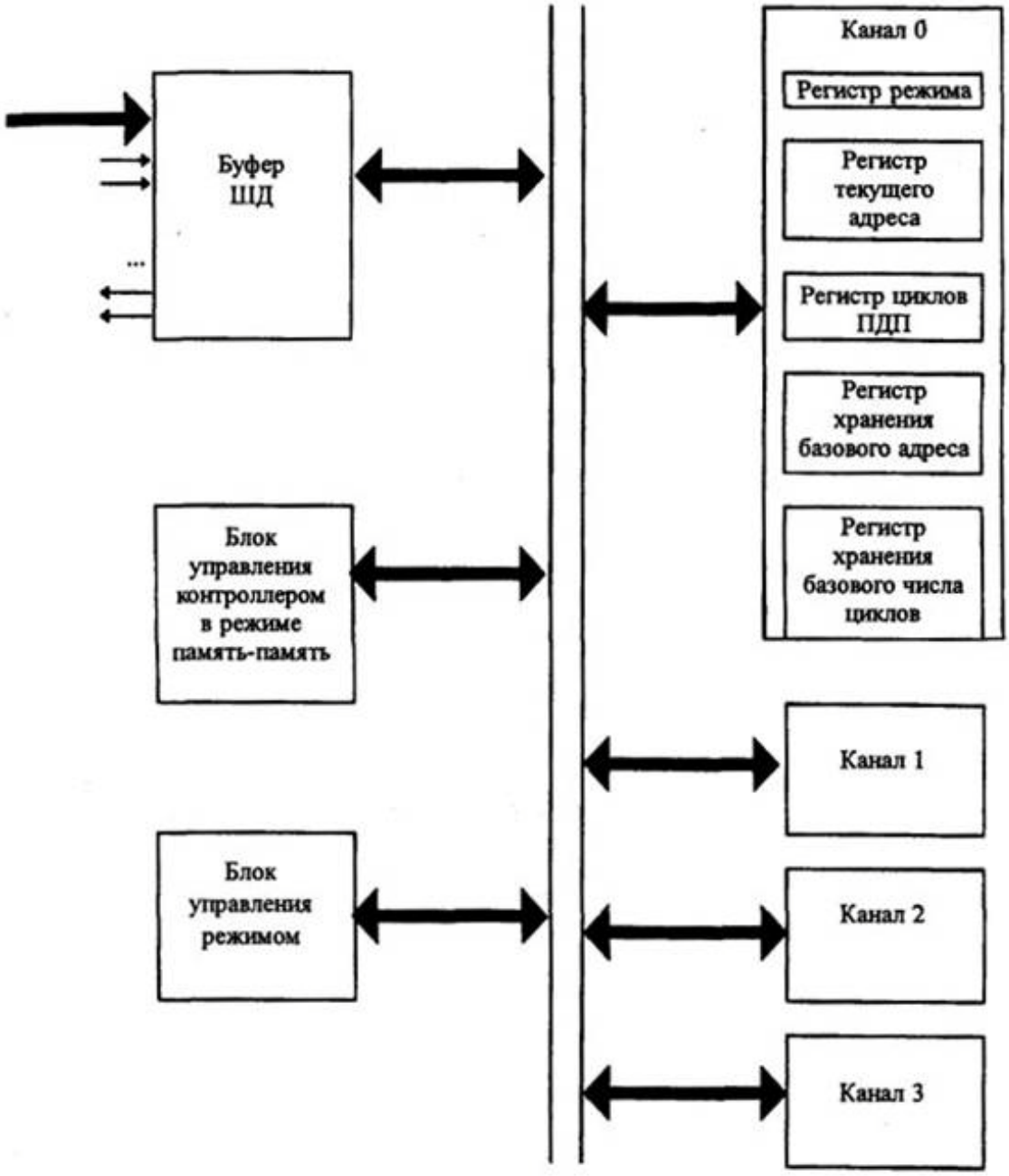


Рис. 6.9. Структурная схема контроллера ПДП

При работе в **режиме прямого доступа к памяти контроллер ПДП выполняет следующие функции:**

- принимает запрос на прямой доступ к памяти от внешнего устройства;
- формирует запрос микропроцессору на захват шин системной магистрали;
- принимает сигнал, подтверждающий вход микропроцессора в состояние захвата (перехода в z-состояние, при котором процессор отключается от системной магистрали);

- формирует сигнал, сообщающий внешнему устройству о начале выполнения циклов ПДП;
- выдает на шину адреса системной магистрали адрес ячейки ОП, предназначенной для обмена;
- вырабатывает сигналы, обеспечивающие управление обменом данными;
- по окончании ПДП контроллер либо организует повторение цикла ПДП, либо прекращает режим ПДП, снимая запросы на него.

Лекция 7. Устройства и принципы управления ЭВМ (продолжение)

Основные вопросы:

7.1. Интерфейсы системной шины и внешних ЗУ

7.2. Способы организации совместной работы периферийных и центральных устройств

7.3. Последовательный и параллельный интерфейсы ввода-вывода

7.1. Интерфейсы системной шины и внешних ЗУ

Системная магистраль является узким местом ЭВМ, так как все устройства, подключенные к ней, конкурируют за возможность передавать свои данные по ее шинам.

Системная магистраль – это среда передачи сигналов управления, адресов, данных, к которой параллельно и одновременно может подключаться несколько компонентов вычислительной системы. Физически системная магистраль представляет собой параллельные проводники на материнской плате, которые называются линиями. Но это еще и алгоритмы, по которым передаются сигналы, правила интерпретации сигналов, дисциплины обслуживания запросов, специальные микросхемы, обеспечивающие эту работу. Весь этот комплекс образует понятие *интерфейс системной магистрали* или *стандарт обмена.*

1. Стандарт **IBM MULTGBUS**

Исторически все интерфейсы СМ ведут свою родословную от стандарта **IBM MULTGBUS**, для которого фирмой был разработан комплект микросхем (chipset). Этот стандарт мог обслуживать передачу 8- и 16-битовых данных, работать в мультипроцессорном режиме с несколькими ведущими устройствами. Понятие “ведущее/ведомое устройство” могло динамически переопределяться с помощью сигналов управления (например, контроллер ПДП в режиме программирования – ведомое устройство, а в активном режиме -ведущее). Для этого стандарта характерно наличие следующих линий: **20 линий адресов, 16 линий данных, 50 управляющих и служебных линий.**

2.Стандарт **Микроканал - MCA**

Для IBM PS-2 разработан стандарт Микроканал - MCA (Micro Channel Architecture) в 1987 г. В нем 24-разрядная шина адреса. Шина данных увеличена до 32 бит. Отказались от перемычек и переключателей, определяющих конфигурацию технических средств, и ввели CMOS-память (Complementary Metal Oxyde Semicondactor), позволяющую хранить эту информацию и при отключении питания. Все оборудование, подключаемое к системной магистрали, содержит специальные регистры POS (Programmable Option Select), позволяющие конфигурировать систему программным путем. При тактовой частоте 10 МГц скорость передачи данных составляла 20 Мбайт/с.

3.Стандарт **ISA**

Для IBM PC XT был разработан **стандарт ISA** (Industry Standart Architecture), который имеет две модификации – для XT и для AT. В ISA XT шина данных – 8 бит, шина адресов – 20 бит, шина управления – 8 линий. В ISA AT

шина данных увеличена до 16 бит. Встречаются и 32-битовые ISA, но это – не стандартизированное расширение. Тактовая частота для работы СМ в стандарте ISA составляет 8 МГц. Производительность ISA XT – 4 Мбайт/с, ISA AT – от 8 до 16 Мбайт/с.

4.Стандарт **EISA**

Стандарт **EISA** (Extended ISA) – это жестко стандартизованное расширение ISA до 32 бит. Конструктивно совместима с ISA-адаптерами внешних устройств. Предназначена для многозадачных систем, файл-серверов и систем, в которых требуется высокоэффективное расширение ввода-вывода. При тактовой частоте 8.33 МГц скорость передачи данных составляла 33 Мбайт/с.

5.Стандарт **VESA**

Стандарт **VESA** (VESA Lokal Bas или VLB) разработан Ассоциацией стандартов видеоданных (Video Electronics Standart Association) как расширение стандарта ISA для обмена видеоданными с адаптером SVGA. Обмен данными по этому стандарту ведется под управлением микросхем, расположенных на карте, устанавливаемой в специальный слот (разъем) расширения VLB и соединяемой с СМ через стандартный слот расширения. В отличие от стандартных слотов расширения слот VLB связан с микропроцессором напрямую, минуя системную магистраль. Карта VLB, работая совместно с системной магистралью, реализующей стандарт ISA, обеспечивает 32-разрядную передачу данных с тактовой частотой микропроцессора (но не более 40–50 МГц). В стандартные слоты материнской платы с интерфейсом VLB устанавливаются карты расширения с интерфейсом ISA. Производительность стандарта VLB достигает 132 Мбайт/с.

6.Стандарт **PCI**

Стандарт **PCI** (Peripheral Component Interconnect) разработан фирмой Intel для ЭВМ с МП Pentium. Это не развитие предыдущих стандартов, а совершенно новая разработка. Системная магистраль в соответствии с этим стандартом работает **синхронно с тактом МП** и осуществляет связь между локальной шиной МП и интерфейсом ISA, EISA или MCA. Но поскольку для этого интерфейса используются микросхемы, выпускаемые другими фирмами (Satum – для 486, Mercury, Neptune, Triton – для Pentium), скорость работы СМ реально'составляет 30 – 40 Мбайт/с при теоретически возможной 132/ 264 Мбайт/с. Стандарт PCI разрабатывался как процессорно-независимый интерфейс. Помимо Pentium с этим интерфейсом могут работать и МП других фирм (Alpha корпорации DEC, MIPS R4400 и Power PC фирм Motorola, Apple и IBM).

Стандарт **PCI** позволяет реализовать дополнительные функции:

- автоматическую конфигурацию периферийных устройств (которая позволяет пользователю устанавливать дополнительные платы, не задумываясь над распределением прерываний, каналов ПДП и адресного пространства);
- работу при пониженном напряжении питания;
- возможность работы с 64-разрядным интерфейсом. "Слоевая" структура интерфейса PCI снижает электрическую нагрузку на МП и позволяет одновременно управлять шестью периферийными

устройствами, подключенными к СМ. Стандарт PCI позволяет использовать "мосты" (Bridges) для организации связи с другими стандартами (например, PCI to ISA Bridge).

7.Стандарт **USB**

Стандарт **USB** (Universal Serial Bus) – универсальный последовательный интерфейс, обеспечивающий обмен со скоростью 12 Мбайт/с и подключение до 127 устройств.

8.Стандарт **PCMCIA**

Стандарт **PCMCIA** (Personal Computer Memory Card International Association) - интерфейс блокнотных ПЭВМ для подключения расширителей памяти, модемов, контроллеров дисков и стриммеров, сетевых адаптеров и др. Системная магистраль, выполненная по этому стандарту, имеет минимальное энергопотребление, ШД – на 16 линий, ША – на 24 линии.

В настоящее время существует множество внешних интерфейсов. Наиболее распространены следующие:

- системная шина (магистраль) *ISA;*
- шина PCI;
- шина AGP;
- шина PC Card (старое название PCMCIA) – обычно только в ноутбуках;
- параллельный порт (принтерный, LPT-порт) Centronic;
- последовательный порт (COM-порт) RS-232C;
- последовательный порт USB (Universal Serial Bus);
- последовательный инфракрасный порт IrDA

Для подключения внешних ЗУ к микропроцессорному комплекту используется один из 5 типов интерфейсов:

1. ST506/412;
2. ESDI (Enhanced Small Device Interface);
3. SCSI (Small Computer System Interface);
4. IDE (Integrated Drive Electronics) известныйтакже, как ATA (AT Attachement);
5. EIDE(Enhanced-IDE).
6. SATA
7. SAS (для серверных станций)

SCSI является промышленным стандартом для подключения таких устройств, как винчестеры, стриммеры, сменные и оптические диски и др.

Этот интерфейс осуществляет параллельную пересылку данных (побайтно) с контролем по четности, что значительно повышает скорость его работы. Применяется не только в IBM-совместимых ЭВМ, но и в VAX, Macintosh, SPARCstation и др. Он обслуживает одновременно до 8 устройств (одним из которых является основной (хост) адаптер SCSI). Хост-адаптер SCSI имеет свою собственную BIOS, которая занимает 16 Кбайт в верхней области памяти (UMB).

Интерфейс обеспечивает удаление внешних ЗУ до 6м при синфазном способе работы и до 25м – при дифференциальном соединении (токовая петля).

Обмен между устройствами на магистрали SCSI происходит в соответствии с протоколом высокого уровня. Программы управления обменом составляются на CCS (Common Command Set) – это универсальный набор команд, обеспечивающих доступ к данным на логическом уровне (в отличие от ESDI).

Программное обеспечение SCSI не оперирует физическими характеристиками жестких дисков (числом цилиндров, головок и т.д.), а имеет дело только с логическими блоками.

Для 32-разрядных микропроцессоров появился интерфейс **SCSI-2**, в спецификацию которого был введен так называемый "широкий" (wide) вариант шины данных – дополнительные 24 линии. Кроме "широкого" был разработан "быстрый" (fast) SCSI-2 с производительностью 10 Мбит/с. Совместное их использование позволяет повысить производительность магистрали до 40 Мбит/с.

Интерфейс может организовывать очередь команд, в нем расширен состав команд. Планируется выпуск **SCSI-3**, позволяющего подключать большее количество устройств и обеспечивающего работу с более длинным кабелем.

Интерфейс **IDE** (он же ATA, AT-bus, PC/AT, Task File) был предложен пользователям AT и XT в 1988 г. в качестве недорогой альтернативы интерфейсам ESDI и SCSI. Его отличительная особенность – реализация функций контроллера в самом накопителе. Такое решение позволяет сократить количество сигналов, передаваемых между системной платой и накопителем (остался один 40-жильный кабель), повысить производительность жесткого диска с 5 до 10 Мбит/с. В контроллере используются такие аппаратные средства, как кэш-память, трансляторы физических параметров диска в логические, что позволяет использовать нестандартные параметры накопителя.

Выпуск малогабаритных компьютеров типа “lap-top” и “notebook” потребовал сокращения размеров, как самого жесткого диска, так и его контроллера, поэтому новая концепция интерфейса IDE стала в них доминирующей.

7.2. Способы организации совместной работы периферийных и центральных устройств

Связь двух ЭВМ и внешнего устройства или двух ЭВМ друг с другом может быть организована в трех режимах: ***симплексном***, ***полудуплексном*** и ***дуплексном***.

В ***симплексном режиме*** передача данных может вестись только в одном направлении: один передает, другой принимает.

Полудуплексный режим позволяет выполнять поочередный обмен данными в обоих направлениях. В каждый момент времени передача может вестись только в одном направлении: один передает, другой принимает. И пока передача не закончилась, принимающий ничего не может сообщить передающему. Заканчивая передачу, передающая ЭВМ пересылает приемной специальный сигнал "перехожу на прием" (или просто "прием" – как выглядит этот сигнал, должны "договориться" между собой коммуникационные программы). Этот сигнал должен быть известен

обеим ЭВМ, т.е. сигнал окончания связи должен выглядеть одинаково у ЭВМ, находящихся на связи. Затем они могут поменяться ролями. Этот режим является самым простым. Если во время передачи в приемной ЭВМ возникла нештатная ситуация (появилась ошибка в передаваемых данных, коммуникационная программа не успела обработать принятый байт до поступления следующего, при распечатке принимаемых данных одновременно с приемом замяло бумагу в принтере и др.), то принимающая ЭВМ неспособна сообщить об этом передающей до появления сигнала окончания передачи. Вся информация, передаваемая после появления нештатной ситуации, теряется. После устранения неполадок передачу приходится повторять. Поэтому при обмене большими объемами информации приходится все передаваемые данные делить на блоки и контролировать прохождение каждого блока. Общее время обмена информацией при этом возрастает.

Дуплексный режим позволяет вести передачу и прием одновременно в двух встречных направлениях.

В симплексном режиме может быть осуществлена связь, например, между ЭВМ и принтером, клавиатурой и ЭВМ, или ЭВМ и дисплеем, а также между двумя ЭВМ, находящимися всегда в односторонней связи.

Для организации симплексного режима необходимо, чтобы передатчик одной ЭВМ был связан с приемником другой ЭВМ двухпроводной линией связи.

Для организации полудуплексного режима можно применить:

- либо специальное коммутационное устройство у каждой ЭВМ, переключающее линию связи с выхода передатчика на вход приемника и обратно,
- либо линию связи с большим количеством проводов (например, трехпроводную, в которой один провод связывает передатчик первой ЭВМ с приемником второй. Другой провод связывает приемник первой ЭВМ с передатчиком второй, а третий является общим проводом и называется *информационная земля).*

Для организации дуплексного режима необходимо, чтобы аппаратурные средства (в состав которых входит и канал связи) обеспечивали возможность одновременной передачи информации во встречных направлениях. Например, дуплексный режим может быть реализован при связи ЭВМ с принтером, если дополнительно к информационной связи канал обеспечивает передачу управляющего сигнала готовности принтера (сигнал DSR).

Сопряжение ЭВМ с каналом связи осуществляется с помощью последовательного (RS-232) или параллельного (Centronics) интерфейса, каждый из которых может обеспечить работу сопрягаемых устройств в любом из рассмотренных режимов - все зависит от типа используемого канала связи и технологии его использования.

Способ, с помощью которого интерфейс обеспечивает связь в заданном режиме, называется *протоколом.* Дуплексная связь ЭВМ с внешним устройством (принтером, модемом), при которой осуществляются симплексный режим обмена информацией, извещение внешнего устройства о готовности ЭВМ с помощью

сигнала DTR и извещение ЭВМ о готовности внешнего устройства с помощью сигнала DSR, обеспечивается *аппаратурным протоколом DTR.*

Программный протокол XON/XOFF основан на использовании программно или аппаратурно-реализуемых сигналов XON (код ASCII 17d или llh) и XOFF (код ASCII 19d или 13h), вырабатываемых принимающим устройством. Эти сигналы имеют направленность, противоположную передаваемому информационному потоку. При получении передающей ЭВМ управляющего кода XOFF она должна прекратить передачу информации до появления разрешающего кода XON.

Управляющие сигналы XON и XOFF передаются по основной информационной линии в дуплексном режиме обмена информацией. Поэтому коммуникационная программа должна постоянно контролировать состояние внешнего устройства (которым может являться и другая ЭВМ), распознавая среди потока информации управляющие сигналы и корректируя в соответствии с ними режим передачи.

Программно-аппаратурный протокол RTS/CTS используется для синхронного обмена информацией (все ранее рассмотренные протоколы реализовали асинхронный обмен) между ЭВМ и ее внешним устройством. В соответствии с этим протоколом производится взаимное оповещение взаимодействующих устройств о выполненных ими действиях: ЭВМ обращается к подключенному внешнему устройству, вырабатывая сигнал DTR (Data Terminal Ready) – "ЭВМ готова к выходу на связь", сопровождающий команду внешнему устройству, находящуюся на информационных линиях интерфейса. Внешнее устройство, получив команду, выполняет ее (обычно первая команда связана с выполнением внешним устройством подготовительных операций – включению, установлению связи, настройке), после чего внешнее устройство выставляет управляющий сигнал DSR (Data Set Ready) – "Внешнее устройство готово", сопровождающий выставленное внешним устройством на информационные линии интерфейса сообщение (например, модем в этом случае выставляет на информационные линии ответный код Connect, информирующий ЭВМ , что связь с другим модемом установлена). Получив сигнал DSR и прочитав ответный код, ЭВМ выставляет сигнал RTS (Request To Send) – "ЭВМ готова к обмену информацией". Внешнее устройство (тот же модем) в ответ на сигнал RTS вырабатывает сигнал CTS (Clear To Send) – "Готов к обмену", по которому коммуникационная программа начинает передачу/прием данных.

Четыре управляющих сигнала: DTR, DSR, RTS, CTS вырабатываются ЭВМ и внешним устройством. Анализ поступивших сигналов производится коммуникационной программой. Передаваемые данные в синхронном режиме могут сопровождаться управляющим сигналом от передающего или приемного устройства (TXD - Transmitted Data и RXD – Received Data соответственно).

В синхронном дуплексном режиме взаимодействующие устройства работают наиболее эффективно, так как выработка большого количества управляющих сигналов позволяет им оперативно информировать друг друга об успешности выполнения каждого шага.

Для взаимодействия со сложными внешними устройствами могут предусматриваться и дополнительные сигналы, например, для модема протокол DTS/CTS содержит- сигналы: DCD (Data Carrier Detected) – "Есть несущая частота" и RI (Ring Indicator) – "Индикатор звонка", информирующий ЭВМ, что по телефонной линии, подключенной к модему, поступили сигналы вызова (звонка), т.е. электрические сигналы, параметры которых отличаются от несущей.

Для того чтобы обеспечить взаимодействие ЭВМ по наиболее сложному протоколу DTS/CTS, последовательный интерфейс RS-232 предусматривает обмен всеми перечисленными сигналами.

Но тот же интерфейс позволяет реализовать обмен и по любому другому протоколу, например, протоколу DTR, для которого в симплексном режиме требуется двух- или трехпроводная линия связи.

7.3. Последовательный и параллельный интерфейсы ввода-вывода

В состав микропроцессорного комплекта входит большая интегральная схема УСАПП (универсальный синхронно-асинхронный приемо-передатчик) или UART (Universal Asynchronous Receiver Transmitter), предназначенная для реализации интерфейса типа RS-232 (V24).

УСАПП является программируемой микросхемой, преобразующей параллельный код, получаемый от шины данных системной магистрали, в последовательный, для передачи по двухпроводной линии связи. В качестве УСАПП используются БИС i8250, П6450, П6550А и др. Функции, выполняемые этими микросхемами, одинаковы. Различия заключаются в обеспечиваемом ими быстродействии.

Типовая структурная схема УСАПП приведена на рис. 7.1.

От микропроцессора передаваемый байт данных поступает по шинам данных (ШД) в буфер данных УСАПП на входной регистр (РгВх), затем через внутреннюю шину передается в регистр передатчика (РгПд). В момент передачи содержимое РгПд серией сдвигов выдвигается в канал с преобразованием в последовательный код.

В синхронном режиме передаваемые данные сопровождаются управляющими сигналами, называемыми *синхронизирующими словами* (СС). Для хранения СС используются специальный регистр РгСС на входе УСАПП и регистр состояния (РгС) - на выходе. Из РгС информация в виде байта состояния передается в микропроцессор по его запросу. Устройство управления (УУ) содержит регистр режима (РгР), предназначенный для хранения передаваемой из микропроцессора информации о режиме работы, и регистр команд (РгК) для хранения принимаемой из микропроцессора команды на обмен данными.

Передаваемый последовательный код перед выходом из передатчика УСАПП в линию связи комплектуется управляющими сигналами, необходимыми для настройки приемника. После такого укомплектования образуется кодовая посылка следующей структуры, изображенной на рис. 7.2.

Старт-бит всегда имеет единичное значение, отличное от состояния "молчащего" канала. Вслед за старт-битом расположены информационные биты,

принятые от шины данных системной магистрали. В зависимости от настройки УСАПП в одной посылке может содержаться от 5 до 8 информационных битов. Значение этих битов в каждой посылке непредсказуемо. В процессе передачи они могут быть искажены помехами. Поэтому в посылке должны содержаться не только биты, говорящие о начале и конце посылки, но и биты для контроля правильности передачи.

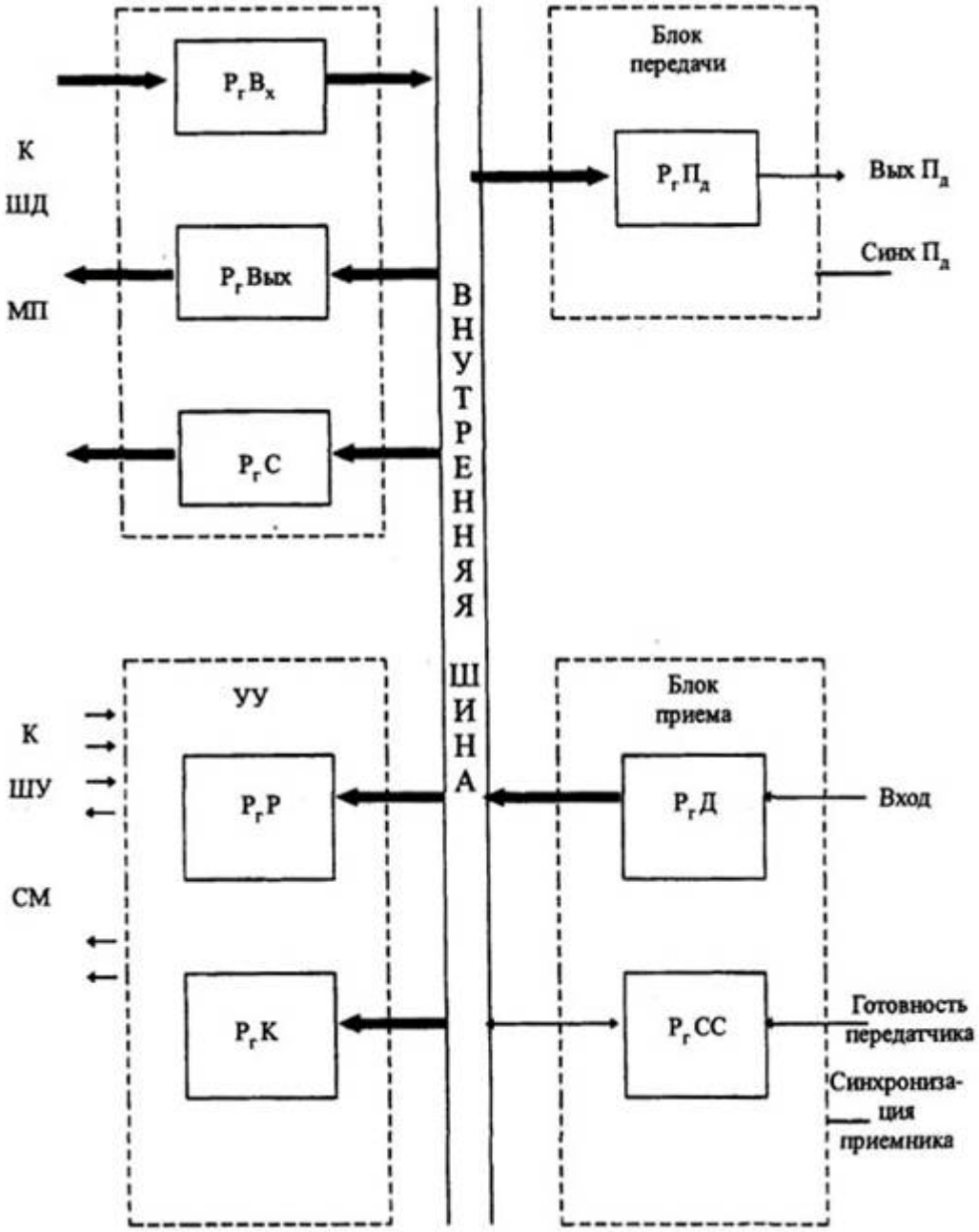


Рис. 7.1. Структурная схема УСАПП

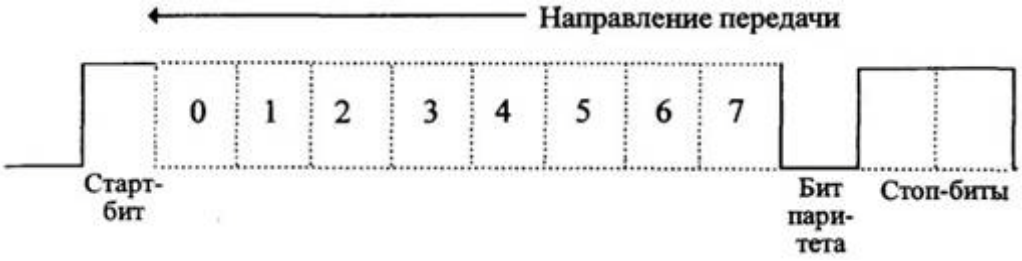


Рис. 7.2. Кодовая посылка УСАПП

В качестве контрольного выступает бит паритета, следующий сразу вслед за информационными битами. С помощью бита паритета осуществляется контроль на четность или нечетность. При контроле на четность сначала подсчитывается количество единиц в информационной части посылки, затем определяется, четное оно или нет. Если полученное число нечетное, бит паритета устанавливается в единицу, в этом случае в правильно переданной посылке всегда будет содержаться четное количество единиц (т.е. единиц, содержащихся в информационных разрядах вместе с битом паритета). При контроле на нечетность бит паритета устанавливается так, чтобы общее количество единиц было всегда нечетным.

При программировании УСАПП программист выбирает: использовать режим контроля или отказаться от него. Он может отказаться от контроля, и бит паритета всегда будет нулевым; может включить контроль на четность или контроль на нечетность. Выбор, что необходимо – контроль на четность или на нечетность, осуществляется в зависимости от характера возможных помех. Если воздействие возможных помех будет проявляться преимущественно в появлении лишних единиц, необходим контроль на четность. Если же воздействие помех будет проявляться преимущественно в исчезновении единиц, то необходим контроль на нечетность (чтобы отличать передаваемый 0 от полной потери информации из-за помех).

После бита паритета в кодовой посылке следуют стоп-биты. Для стоп-битов в кодовой посылке отводятся два двоичных знакоместа. Если выбран режим "1 стоп-бит", то после бита паритета всегда (в каждой посылке) будет следовать комбинация 01. Если выбран режим "1,5 стоп-бита", то после бита паритета всегда будет следовать комбинация 10. Если же выбирается режим "2 стоп-бита", то каждая посылка будет завершаться цифрами 11.

В УСАПП-приемнике поступившая от канала связи кодовая комбинация проверяется в соответствии с установленным заранее режимом контроля (на четность или нечетность), освобождается от управляющих сигналов и передается в шину данных системной магистрали параллельным кодом.

Настраиваться УСАПП-приемник и УСАПП-передатчик, работающие в паре, должны согласованно.

Программирование УСАПП может вестись на физическом или логическом уровне. Программирование на физическом уровне производится на языках низкого уровня или в машинных кодах. Логический уровень программирования обеспечивается алгоритмическими языками высокого уровня, коммуникационными программами, некоторыми пакетами прикладных программ.

Параллельный интерфейс представлен в микропроцессорном комплекте микросхемой типа i8255 – контроллером параллельного интерфейса или программируемым интерфейсным адаптером.

Микросхема подключается к системной магистрали ЭВМ (соответственно – к шинам данных, адреса и управления) и имеет три независимых канала для подключения внешних устройств. Внутренний блок управления позволяет программировать каждый канал на ввод или вывод информации по 8 линиям, т.е. 8 бит параллельно.

Лекция 8. Концепция GRID – технологии, метакомпьютинг и облачные вычисления

Основные вопросы:

8.1. Введение
8.2. Концепция GRID – технологии
8.3. Понятие метакомпьютинга
8.4. Облачные вычисления
8.5. Заключение

8.1. Введение

Рассматриваемую тему можно обозначить как использование компьютерных сетей для создания распределенной вычислительной инфраструктуры национального и мирового масштаба. На сегодня сети доказали беспрецедентную практическую полезность, выступая как средство глобальной доставки различных форм информации.

Рис. 8.1. Общее представление о GRID-технологии

Internet представляет собой множество узлов с собственными процессорами, оперативной и внешней памятью, устройствами ввода/вывода. Узлы соединены друг с другом коммутационным оборудованием и линиями передачи данных. Такая конструкция весьма напоминает многопроцессорную систему, в которой роль магистральных шин выполняет Сеть.

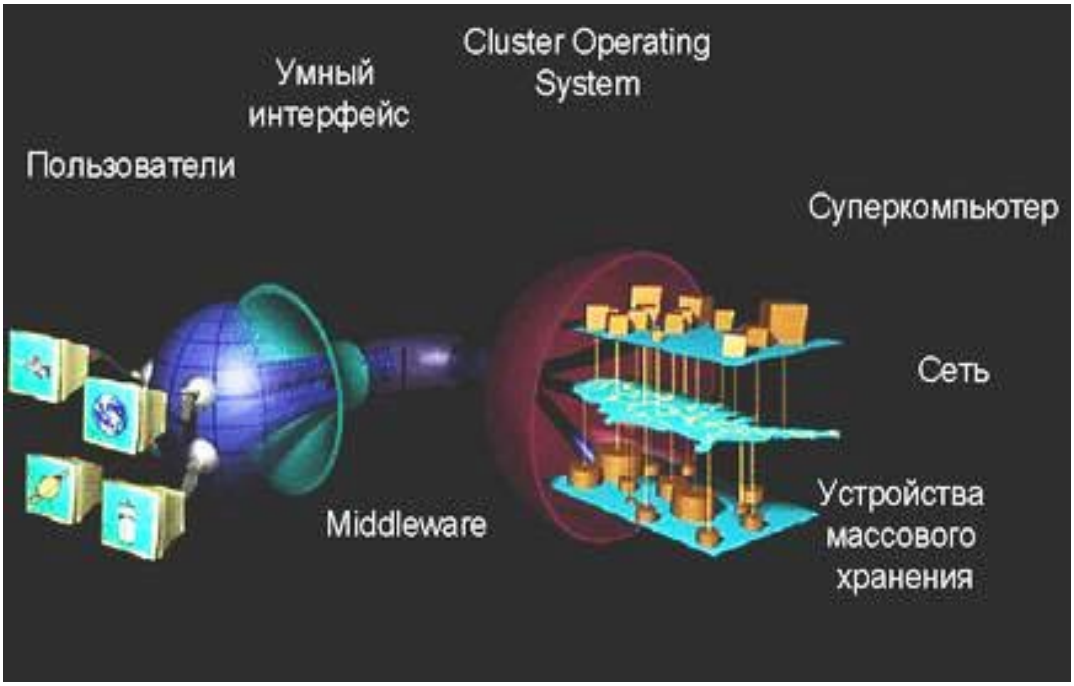


Рис. 8.2. Участники GRID-вычислений

Цель заключается в том, чтобы превратить аналогии в реальность, то есть стереть барьеры между разнородными, пространственно распределенными вычислительными системами, образовав сверхкомпьютер или метакомпьютер, который для пользователей и программистов выступал бы как единая вычислительная среда, доступная непосредственно с рабочего места (ПК или рабочей станции).

Центральное понятие метакомпьютера можно определить, как метафору виртуального компьютера, динамически организующегося из географически распределенных ресурсов, соединенных высокоскоростными сетями передачи данных. Необходимо подчеркнуть принципиальную разницу метакомпьютерного подхода и сегодняшних программных средств удаленного доступа. В метакомпьютере этот доступ прозрачен, то есть пользователь имеет полную иллюзию использования одной, но гораздо более мощной, чем та, что стоит на его столе, машины и может с ней работать в рамках той же модели, которая принята на его персональном вычислителе.

Зачем вообще может быть нужна такая среда? Непосредственные потребности исходят от высокопроизводительных приложений. В различных прикладных областях (космологии, гидрологии окружающей среды, молекулярной биологии и т.д.) поставлены весьма важные задачи, характеризующиеся, например, следующими требованиями к компьютерным ресурсам:

- 0.2 — 20 Tflops процессорной мощности;
- 100 — 200 GB оперативной памяти;
- 1— 2 TB дисковой памяти;
- 0.2 — 0.5 GB/sec ширина полосы пропускания ввода/вывода.

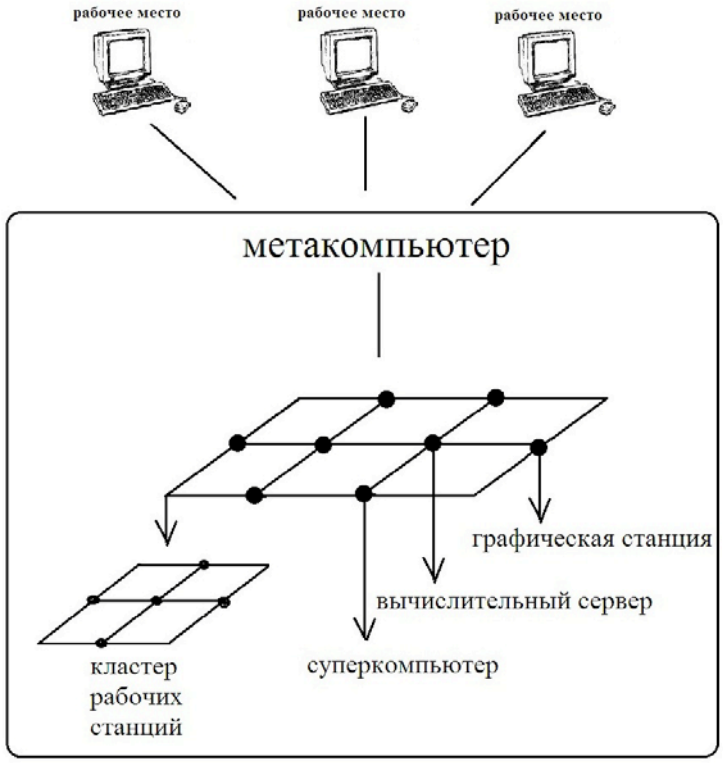


Рис. 8.3. Типы узлов метакомпьютера

Нижняя граница таких запросов — это уникальные архитектуры типа SGI/CRAY Origin с тысячами процессоров. С другой стороны, суммарный объем ресурсов в достаточно большом фрагменте Сети далеко превосходит эти цифры, вопрос в том, как эти ресурсы объединить и дать в руки реальному потребителю.

В результате суперкомпьютерные мощности стали доступны практически всем заинтересованным исследователям, произошла быстрая эволюция архитектур: от векторных систем (PVP) к машинам с массовым параллелизмом (MPP) и далее к машинам с симметричным мультипроцессированием на базе разделяемой памяти (SMP).

8.2. Концепция GRID – технологии

Новая интернет-технология, получившая название GRID-computing (дословно - решеточные вычисления), считается большинством исследователей следующим шагом в развитии Интернета. Концепцию GRID-технологии можно рассматривать как концепцию глобальной инфраструктуры, интегрирующей мировые компьютерные ресурсы для реализации крупномасштабных информационно-вычислительных проектов.

Термин "GRID" был предложен Яном Фостером (Ian Foster) и Карлом Кессельманом (Karl Kesselman), создателями первой книжки об идее использования компьютерных сетей для решения задач, требующих особо огромных вычислительных ресурсов, и разъясняется некой аналогией с электрическими сетями (Power Grid). Power Grid - сеть электропитания, распределенный ресурс общего использования, построенная таким образом, что каждый может просто подключиться через розетку и взять электричества, сколько ему требуется, не задумываясь о том, откуда это электричество "пришло".

GRID – географически распределенная инфраструктура, объединяющая множество ресурсов разных типов (процессоры, долговременная и оперативная память, хранилища и базы данных, сети), доступ к которым пользователь может получить из любой точки, независимо от места их расположения. GRID предполагает коллективный разделяемый режим доступа к ресурсам и к связанным с ними услугам в рамках глобально распределенных виртуальных организаций, состоящих из предприятий и отдельных специалистов, совместно использующих общие ресурсы. В каждой виртуальной организации имеется своя собственная политика поведения ее участников, которые должны соблюдать установленные правила. Виртуальная организация может образовываться динамически и иметь ограниченное время существования.

Потенциал технологий GRID уже сейчас оценивается очень высоко: он имеет стратегический характер, и в близкой перспективе должен стать вычислительным инструментарием для развития высоких технологий в различных сферах человеческой деятельности, подобно тому, как подобным инструментарием стали персональный компьютер и интернет. Такие высокие оценки можно объяснить способностью GRID на основе безопасного и надежного удаленного доступа к ресурсам глобально распределенной инфраструктуры решить две проблемы:

- создания распределенных вычислительных систем сверхвысокой пропускной способности из серийно выпускаемого оборудования (показатели производительности: агрегированная мощность более 1 терафлоп, объем обрабатываемых данных более 1 петабайта в год) при одновременном повышении эффективности (до 100%) имеющегося парка вычислительной техники путем предоставления в грид временно простаивающих ресурсов;
- создания широкомасштабных систем мониторинга, управления, комплексного анализа и обслуживания с глобально распределенными источниками данных, способных поддерживать жизнедеятельность государственных структур, организаций и корпораций.

Рассмотрим области применения GRID**:**

Изначально GRID - технологии предназначались для решения сложных научных, производственных и инженерных задач, которые невозможно решить в разумные сроки на отдельных вычислительных установках. Однако теперь область применения технологий GRID не ограничивается только этими типами задач. По мере своего развития GRID проникает в промышленность и бизнес, крупные предприятия создают GRID для решения собственных производственных задач. Таким образом, GRID претендует на роль универсальной инфраструктуры для обработки данных, в которой функционирует множество служб (Grid Services), которые позволяют решать не только конкретные прикладные задачи, но и предлагают сервисные услуги: поиск необходимых ресурсов, сбор информации о состоянии ресурсов, хранение и доставка данных. Применение GRID может дать новое качество решения следующих классов задач:

- массовая обработка потоков данных большого объема;
- многопараметрический анализ данных;
- моделирование на удаленных суперкомпьютерах;

- реалистичная визуализация больших наборов данных;
- сложные бизнес-приложения с большими объемами вычислений.

Технологии GRID включают в себя лишь наиболее общие и универсальные аспекты, одинаковые для любой системы (архитектура, протоколы, интерфейсы, сервисы). Используя эти технологии и наполняя их конкретным содержанием, можно реализовать ту или иную GRID-инфраструктуру, предназначенную для решения того или иного класса прикладных задач. GRID-технологии не являются технологиями параллельных вычислений. Их основная задача - координация использования ресурсов. Хотя в рамках конкретной GRID-системы возможно организовать параллельные вычисления с использованием существующих параллельных технологий.

Для построения полностью функциональной грид-системы необходимо программное обеспечение промежуточного уровня (middleware), построенное на базе существующих инструментальных средств и предоставляющее высокоуровневые сервисы задачам и пользователям. Создание и реализация GRID-технологий являются сложной научной и практической проблемой, находящейся на стыке большого количества научно-технических направлений.

GRID-технологии уже активно применяются как государственными организациями управления, обороны, сферы коммунальных услуг, так и частными компаниями, например, финансовыми и энергетическими. Область применения GRID сейчас охватывает ядерную физику, защиту окружающей среды, предсказание погоды и моделирование климатических изменений, численное моделирование в машино- и авиастроении, биологическое моделирование, фармацевтику.

8.3. Понятие метакомпьютинга

Термин «метакомпьютинг» возник вместе с развитием высокоскоростной сетевой инфраструктуры в начале 90-х годов и относился к объединению нескольких разнородных вычислительных ресурсов в локальной сети организации для решения одной задачи. Основная цель построения мета-компьютера в то время заключалась в оптимальном распределении частей работы по вычислительным системам различной архитектуры и различной мощности. Например, предварительная обработка данных и генерация сеток для счета могли производится на пользовательской рабочей станции, основное моделирование на векторно-конвейерном суперкомпьютере, решение больших систем линейных уравнений – на массивно-параллельной системе, а визуализация результатов – на специальной графической станции.

В дальнейшем, исследования в области технологий метакомпьютинга были развиты в сторону однородного доступа к вычислительным ресурсам большого числа (вплоть до нескольких тысяч) компьютеров в локальной или глобальной сети. Компонентами "мета-компьютера" могут быть как простейшие ПК, так и мощные массивно-параллельные системы. Что важно, мета-компьютер может не иметь постоянной конфигурации – отдельные компоненты могут включаться в его конфигурацию или отключаться от нее; при этом технологии метакомпьютинга обеспечивают непрерывное функционирование системы в целом. Современные исследовательские проекты в этой области направлены на обеспечение

прозрачного доступа пользователей через Интернет к необходимым распределенным вычислительным ресурсам, а также прозрачного подключения простаивающих вычислительных систем к мета-компьютерам.

Очевидно, что наилучшим образом для решения на мета-компьютерах подходят задачи переборного и поискового типа, где вычислительные узлы практически не взаимодействуют друг с другом и основную часть работы производят в автономном режиме. Основная схема работы в этом случае примерно такая: специальный агент, расположенный на вычислительном узле (компьютере пользователя), определяет факт простоя этого компьютера, соединяется с управляющим узлом мета-компьютера и получает от него очередную порцию работы (область в пространстве перебора). По окончании счета по данной порции вычислительный узел передает обратно отчет о фактически проделанном переборе или сигнал о достижении цели поиска.

Понятие **метакомпьютера** можно определить, как метафору виртуального компьютера, динамически организующегося из географически распределенных ресурсов, соединенных высокоскоростными сетями передачи данных. Отдельные установки являются составными частями метакомпьютера и в то же время служат точками подключения пользователей.

Распространение метакомпьютерных технологий может произойти только при гармоничном сочетании двух направлений: развития технической базы и создания программного обеспечения нового поколения.

Формы метакомпьютера

1**. Настольный суперкомпьютер**. Пользователь получает возможность запускать свои задачи на удаленных вычислительных установках с таким объемом вычислительных ресурсов, которые необходимы для успешного счета. При этом от пользователя не требуется искать подходящие не занятые мощности: распределять задачи в сети в соответствии с их запросами — функция метакомпьютера. В рамках метакомпьютинга разрабатываются схемы глобальных прав доступа, дающие возможность пользователю вступать во временное владение найденными ресурсами без персональной регистрации на исполнительной установке и одновременно гарантирующие надежную защиту.

Кроме доступа к многопроцессорным комплексам для решения распараллеленных задач, этот режим полезен и для выхода на ресурсы других типов, например, на мощные графические станции со специализированными процессорами или на базы данных с большими объемами информации, которые по тем или иным причинам не могут быть тиражированы (рис.8.3).

Переход к работе в режиме настольного суперкомпьютинга достаточно безболезнен как для пользователей, так и для программистов. Фактически они работают со штатными ОС исполнительных установок. Этот режим, однако, представляется весьма важным этапом, открывающим реальные возможности освоения передовых компьютерных технологий всему научному и инженерному сообществу, независимо от расположения рабочих мест пользователей и необходимых им вычислительных ресурсов. Как результат, можно ожидать более эффективного использования уникального и дорогостоящего оборудования,

резкого роста числа приложений, в которых находят применение самые передовые методы обработки данных.

2**. Интеллектуальный инструментальный комплекс**. Практический опыт из многих прикладных областей показывает, что быстро считать недостаточно: часто необходимо в реальном времени собирать большие объемы данных, поступающих с датчиков, производить анализ текущей ситуации, вырабатывать решения и выдавать управляющие воздействия. Все это требует тесной интеграции управления, обработки данных разного вида, моделирования процессов, визуализации в реальном времени. Вычислительные комплексы такого рода получили название интеллектуальных инструментов.

3. **Сетевой суперкомпьютер**. При таком подходе идея метакомпьютинга доводится до логической завершенности, а именно: масштабирование всех возможных вычислительных ресурсов путем прозрачного бесшовного объединения посредством сети отдельных вычислительных установок разной мощности. Составляющими элементами такой конструкции могут быть суперкомпьютеры, серверы, рабочие станции и даже персональные компьютеры. Отличительной особенностью этой формы является то, что суммарные ресурсы агрегированной архитектуры могут быть использованы в рамках одной задачи.

В такой форме можно выделить два уровня. Первый применим как альтернатива суперкомпьютеру "в ящике". Сетевой суперкомпьютер создается из относительно недорогих компонентов (серверов — рабочих станций) путем их соединения локальной сетью. Известно, что стоимость такого решения даже на базе дорогого и мощного сетевого оборудования все же на порядок меньше цены готового суперкомпьютера (а это миллионы долларов) при сопоставимых характеристиках процессорной производительности.

Второй уровень — для тех, кто имеет суперузлы, то есть настоящие или сетевые суперкомпьютеры. Объединение их в региональном и национальном масштабах скоростными глобальными линиями связи способно дать беспрецедентные мощности. Собственно, этот уровень масштабирования точно соответствует термину метакомпьютинг.

Сопоставляя приведенные четыре формы метакомпьютинга, следует сказать, что полезность их в конкретных условиях в большой степени определяется степенью развитости сетевой инфраструктуры и наличием (или отсутствием) высокопроизводительной техники. Представляется, что в наших условиях (отсутствие суперкомпьютеров и качественных линий связи) ценность метакомпьютерного подхода не только не уменьшается, а напротив, возрастает, нужны только правильные и достижимые приоритеты. Правильная последовательность шагов видится следующим образом:

- обеспечение дистанционного доступа к крупным корпоративным вычислительным центрам (настольный суперкомпьютинг);
- создание единой вычислительной среды в тех же центрах с помощью локальных сетей (сетевой суперкомпьютер);
- по мере развития аппаратной инфраструктуры агрегация вычислительных центров в региональном и далее, в национальном масштабе (интеллектуальные инструменты и метакомпьютер).

Далее вкратце описаны и приведены ссылки на основные исследовательские проекты в области метакомпьютинга, разработанные программные технологии, конкретные примеры метакомпьютеров.

Наиболее известные проекты по метакомпьютингу и распределенным вычислениям в Интернет:

- **Distributed.net**

  Одно из самых больших объединений пользователей Интернет, предоставляющих свои компьютеры для решения крупных переборных задач. Основные проекты связаны с задачами взлома шифров (RSA Challenges). В частности, 19 января 1999 года была решена предложенная RSA Data Security задача расшифровки фразы, закодированной с помощью шифра DES-III. В настоящее время в distributed.net идет работа по расшифровке фразы, закодированной с 64-битным ключом (RC5-64). С момента начала проекта в нем зарегистрировались 191 тыс. человек. Достигнута скорость перебора, равная 75 млрд. ключей в секунду (всего требуется проверить 264 ключей). За решение этой задачи RSA предлагает приз в $10 тыс.

- **GIMPS - Great Internet Mersenne Prime Search**

  Поиск простых чисел Мерсенна (т.е. простых чисел вида $\mathbf{2^P-1}$). С начала проекта было найдено 4 таких простых числа. Организация Electronic Frontier Foundation предлагает приз в $100 тыс. за нахождение простого числа Мерсенна с числом цифр 10 млн.

- **SETI@home**

  Проект SETI@home (Search for Extraterrestrial Intelligence) – поиск внеземных цивилизаций с помощью распределенной обработки данных, поступающих с радиотелескопа. Присоединится может любой желающий. Доступны клиентские программы для Windows, Mac, UNIX, OS/2 (клиент Windows срабатывает в качестве screen-saver'a). Для участия в проекте зарегистрировались около 920 тыс. человек. (На сегодня проект закрыт, на его основе разработано несколько новых).

- **Проект EU Data GRID (EDG)**

  Крупный европейский проекта EU Data Grid (http://www.eu-datagrid.org) для физики высоких энергий, биоинформатики и системы наблюдений за Землей. Общим во всех этих исследованиях является разделение данных по различным базам, распределенным по всем континентам. Основная их цель — улучшение эффективности и скорости анализа данных посредством интеграции глобально распределенных процессорных мощностей и систем хранения данных, доступ к которым будет характеризоваться динамическим распределением по GRID-инфрастуктуре, что предполагает управление репликацией и кэшированием. Проект включает в себя несколько рабочих пакетов:

  - ✓ создание для всех рассматриваемых отраслей (физики высоких энергий, биологии и наблюдения Земли) приложений, осуществляющих прозрачный доступ к распределенным данным и высокопроизводительным вычислительным ресурсам;

- ✓ управление рабочей загрузкой (распределенное планирование и управление ресурсами);
- ✓ управление данными (создание интегрированного инструментария и инфраструктуры промежуточного слоя для согласованного управления и разделения петабайтных объемов данных с эффективным использованием ресурсов);
- ✓ мониторинг (доступ к информации о состоянии и об ошибках в grid-инфраструктуре);
- ✓ управление кластерами, состоящими из тысяч вычислительных узлов;
- ✓ создание виртуальной частной сети, объединяющей вычислительные ресурсы и ресурсы данных, участвующие в отладке grid-инфраструктуры;
- ✓ управление массовой памятью (создание глобального grid-интерфейса к существующим системам управления массовой памятью).

- **Globus**

  Разработка ПО для организации распределенных вычислений в Интернет. Проект реализуется в Argonne National Lab. Цель - построение "computational grids", включающие вычислительные системы, системы визуализации, экспериментальные установки. В рамках проекта проводятся исследовании по построению распределенных алгоритмов, обеспечению безопасности и отказоустойчивости мета-компьютеров.
- **The Metacomputing Project**

  Совместный проект Oak Ridge National Labs, Sandia National Labs и Pittsburgh Supercomputer Center В рамках данного проекта объединены 4 системы: три суперкомпьютера Intel XP/S 150 Paragon и суперкомпьютер T3D в PSC. Коммуникации обеспечиваются с помощью PVM. Для связи используются высокоскоростные сети ESNet и vBNS и протоколы ATM/AAL5.
- **PACX-MPI**

  Расширение MPI для поддержки распределенных вычислений. Поддерживается объединение в единый мета-компьютер нескольких MPP-систем, возможно с различными реализациями MPI. Передача данных между MPP производится через Интернет с помощью TCP/IP. В настоящее время в рамках PACX-MPI реализовано подмножество стандарта MPI 1.2.
- **Condor**

  Система Condor разрабатывается в университете шт. Висконсин (Madison). Condor распределяет независимые подзадачи по существующей в организации сети рабочих станций, заставляя компьютеры работать в свободное время (то есть в то время, когда они простаивали бы без своих пользователей). Программное обеспечение системы Condor доступно бесплатно. В настоящее время поддерживаются платформы SGI, Solaris, Linux, HP-UX, и Digital Unix, однако планируется также поддержка Windows NT.

8.4. Облачные вычисления

Облачные вычисления (англ. *cloud computing*), в информатике – это модель обеспечения повсеместного и удобного сетевого доступа по требованию к общему пулу (англ. *pool*) конфигурируемых вычислительных ресурсов (например, сетям передачи данных, серверам, устройствам хранения данных, приложениям и сервисам— как вместе, так и по отдельности), которые могут быть оперативно предоставлены и освобождены с минимальными эксплуатационными затратами и/или обращениями к провайдеру.

Потребители облачных вычислений могут значительно уменьшить расходы на инфраструктуру информационных технологий (в краткосрочном и среднесрочном планах) и гибко реагировать на изменения вычислительных потребностей, используя свойства вычислительной эластичности (англ. *elastic computing*) облачных услуг.

По оценке IDC рынок публичных облачных вычислений в 2009 году составил $17 млрд - около 5 % от всего рынка информационных технологий.

Суть концепции облачных вычислений заключается в предоставлении конечным пользователям удаленного динамического доступа к услугам, вычислительным ресурсам и приложениям (включая операционные системы и инфраструктуру) через интернет. Развитие сферы хостинга было обусловлено возникшей потребностью в программном обеспечении и цифровых услугах, которыми можно было бы управлять изнутри, но которые были бы при этом более экономичными и эффективными за счет экономии на масштабе.

Большинство сервис-провайдеров предлагают облачные вычисления в формеVPS-хостинга, виртуального хостинга, и ПО-как-услуга (SaaS). Облачные услуги долгое время предоставлялись в форме SaaS, например, Microsoft Hosted Exchange и SharePoint.

Вычислительные облака состоят из тысяч серверов, размещенных в дата-центрах, обеспечивающих работу десятков тысяч приложений, которые одновременно используют миллионы пользователей. Непременным условием эффективного управления такой крупномасштабной инфраструктурой является максимально полная автоматизация. Кроме того, для обеспечения различным видам пользователей – облачным операторам, сервис-провайдерам, посредникам, ИТ-администраторам, пользователям приложений – защищенного доступа к вычислительным ресурсам облачная инфраструктура должна предусматривать возможность самоуправления и делегирования полномочий.

Концепция облачных вычислений значительно изменила традиционный подход к доставке, управлению и интеграции приложений. По сравнению с традиционным подходом, облачные вычисления позволяют управлять более крупными инфраструктурами, обслуживать различные группы пользователей в пределах одного облака, а также означают полную зависимость от провайдера облачных услуг.

Облачные вычисления – это эффективный инструмент повышения прибыли и расширения каналов продаж для независимых производителей программного обеспечения (ISV), операторов связи и VAR-посредников (в форме SaaS). Этот подход позволяет организовать динамическое предоставление услуг, когда

пользователи могут производить оплату по факту и регулировать объем своих ресурсов в зависимости от реальных потребностей без долгосрочных обязательств.

Для хостеров облачные вычисления обеспечивают огромный потенциал роста. Индустрия облачных вычислений стремительно развивается и, по прогнозам аналитиков, к 2014 году на ее долю будет приходиться порядка 10% всех расходов на ИТ. Кроме того, акценты в отрасли все больше смещаются от хостинга к облачным вычислениям и SaaS.

Существует мнение, что в ближайшие 5-10 лет бо́льшая часть ИТ переместится в облака пяти различных типов. Будут проприетарные платформенные облака, предоставляющие различные платформенные услуги, – Google (тип 1), Microsoft (тип 2) и другие крупные ИТ игроки (тип 3), такие как IBM, Apple, HP и Amazon.

Будут облака услуг (тип 4), где ожидается возникновение тысяч облачных провайдеров, предлагающих широкий спектр услуг. В качестве примера можно привести веб-хостинг и хостинг приложений, вертикально интегрированные структуры (правительство, здравоохранение, и т.д.), независимых производителей ПО (стратегическое развитие бизнеса, системы клиентской поддержки и т.д.), телекоммуникационные услуги (голосовая почта, VOIP). И наконец будут облака, управляемые корпоративными ИТ (тип 5), которые будут предоставлять услуги для внутреннего использования и для использования сотрудниками и партнерами.

Компания Forrester Research опубликовала в апреле 2011 года прогноз развития рынка публичных облачных вычислений до 2020 г. Согласно сведениям отчета, к 2020 г. объем облачного рынка составит $241 млрд, что на $200 млрд больше, чем в 2011 г.

Встает вопрос: когда логично использовать облачные технологии?

- Когда контент, генерируемый пользователями, является частью приложения. Если приложение будет принимать файлы или сгенерированный контент будет храниться в файловой системе, рано или поздно придётся задуматься об облачном хранении данных.
- Когда приложению тесно на одном сервере. Можно горизонтально масштабировать сервис на неограниченное количество серверов приложений без необходимости реплицировать файловую систему активов на новых серверах. Так как активы будут храниться централизованно, они будут досягаемы с любого места, вне зависимости от количества серверов, на которых будет работать приложение.
- Когда разработка программы является критической для бизнеса, нежели разработка масштабируемой файловой системы. Если мало либо денег, либо времени и предвидится рост приложения, облачное хранилище будет беспроигрышным вариантом. Оно даст возможность быстро развёртывать приложение и масштабировать его по мере необходимости.

Достоинства облачных вычислений:

- отказоустойчивость;
- безопасность;

- высокая скорость обработки данных;
- снижение затрат на аппаратное и программное обеспечение, на обслуживание и электроэнергию;
- экономия дискового пространства (и данные, и программы хранятся в интернете).

Недостатки облачных вычислений:

- зависимость сохранности пользовательских данных от компаний, предоставляющих услугу cloud computing;
- появление новых («облачных») монополистов.

8.5. Заключение

Общее соображение, обосновывающее обще значимость технологий, развиваемых в рамках метакомпьютерного направления, состоит в том, что современные ОС во все большей степени становятся, завязаны на Сеть, ориентируясь на коллективные и мобильные формы работы пользователей с прямым доступом к общей информации и ПО. Это определяет общую направленность движения.

Во-вторых, на массовом рынке наметилась тенденция в сторону параллелизации и серийно выпускаемых архитектур (рядовыми стали многопроцессорные RISC-серверы и ПК с 4–8 процессорами Intel), и ОС, и работающих в них приложений. Все ведущие производители, выпуская свои ОС, делают одну версию для всего ряда машин – от рабочих станций до суперкомпьютеров, так что фактически даже на простейших однопроцессорных компьютерах есть как базовая, так и инструментальная поддержка параллельных вычислений.

Метакомпьютерный подход, ставя проблему создания полной среды и инфраструктуры для сетевых вычислений, определяет один из возможных контекстов, в котором перечисленные методы могут занять свое место.

Конкретно, интерес представляют работы по:

- глобальным файловым системам,
- системам сертификации и авторизации пользователей,
- оптимизации сетевой передачи данных,
- управлению ресурсами, планированию и диспетчеризации процессов.

Сейчас условия для участия в общем движении в области метакомпьютинга достаточно благоприятны – тексты разработанного ПО обычно открыты, есть оперативный доступ по WWW к планам разработчиков и документации, можно принимать участие в обсуждении проблем по электронной почте.

P.S. См. презентации по теме из папки «Дополнительные материалы»