Министерство образования Российской Федерации

Волгоградский государственный технический университет

Е.И. Духнич , А.Е. Андреев

ОРГАНИЗАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ МАШИН И СИСТЕМ

Учебное пособие

Волгоград
2003

УДК 681.3

Рецензенты:

Духнич Е.И., Андреев А.Е., Организация вычислительных машин и систем: Учебн. пособие/ВолгГТУ, Волгоград, 2003.-80с.

В пособии рассматриваются основные архитектурные особенности построения современных вычислительных машин, систем памяти, процессоров, организация многопроцессорных и специализированных вычислительных систем, основные характеристики ВМ, приводятся классификации вычислительных машин и систем.

Предназначено для студентов дневного и вечернего отделений направления «Информатика и вычислительная техника».

Ил. 34. Табл. 6. Библиогр.: 16 названий.

Печатается по решению редакционно-издательского совета Волгоградского государственного технического университета

ОГЛАВЛЕНИЕ:

1. ОБЩАЯ ХАРАКТЕРИСТИКА И КЛАССИФИКАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ МАШИН И СИСТЕМ

1.1. Аппаратные и программные средства реализации алгоритмов

Вычислительная машина (ВМ) - это искусственная инженерная система для автоматической обработки информации по заданному алгоритму.

Как известно, средства реализации алгоритмов вычислений делятся на аппаратные и программные. Любая вычислительная структура (ВС) это совокупность указанных средств. Их соотношение определяется требованиями к производительности и стоимости ВС.

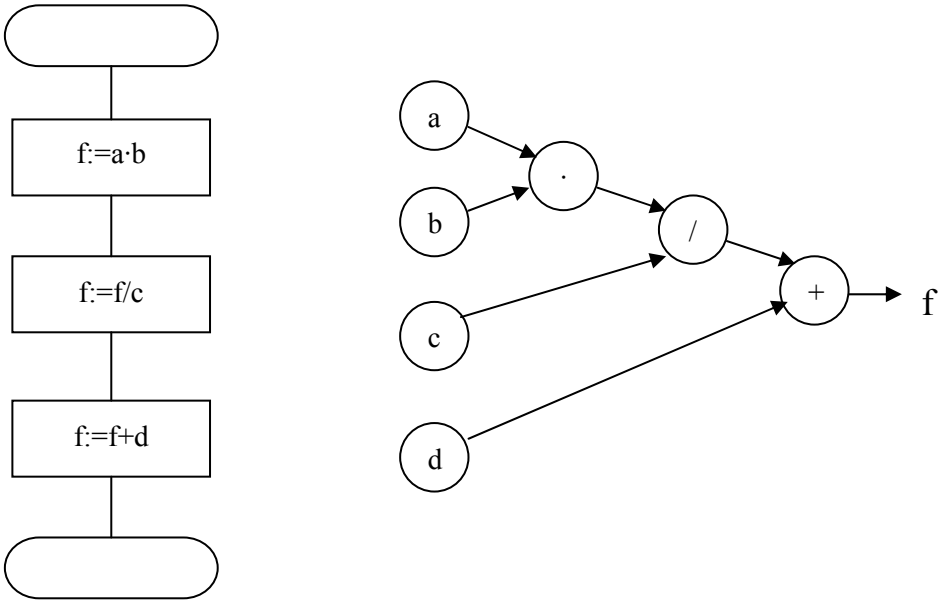
Аппаратные средства реализуют какие-либо действия алгоритма одномоментно, без возможности дробления со стороны программиста. (Примеры аппаратной реализации: сумматоры, быстрые умножители, устройства для преобразования сигналов в реальном времени и т.д.)

Программные средства – это совокупности инструкций по реализации вычислительного процесса с помощью аппаратных средств в соответствии с алгоритмом. Традиционно под программированием обычно понимают процедурное программирование – задание последовательности действий по реализации алгоритма, причем действия происходят последовательно во времени. В то же время «программировать» решение задачи можно и структурно, пользуясь заданным набором аппаратных средств, в этом случае программирование – это указание путей следования потоков данных от одних аппаратных средств к другим. (Термин «структурное программирование» в литературе по вычислительной технике обычно используется для указания на определенную методологию разработки программного обеспечения, подразумевающую нисходящее проектирование системы, использование только основных управляющих конструкций, отказ от операторов GOTO и т.д. В данном контексте «структурное программирование» означает программирование в пространстве аппаратных структур.) Структурное программирование еще называют «аппаратурно-ориентированным».

Программирование структуры и процедурное программирование не являются взаимоисключающими подходами, как правило, они дополняют друг друга.

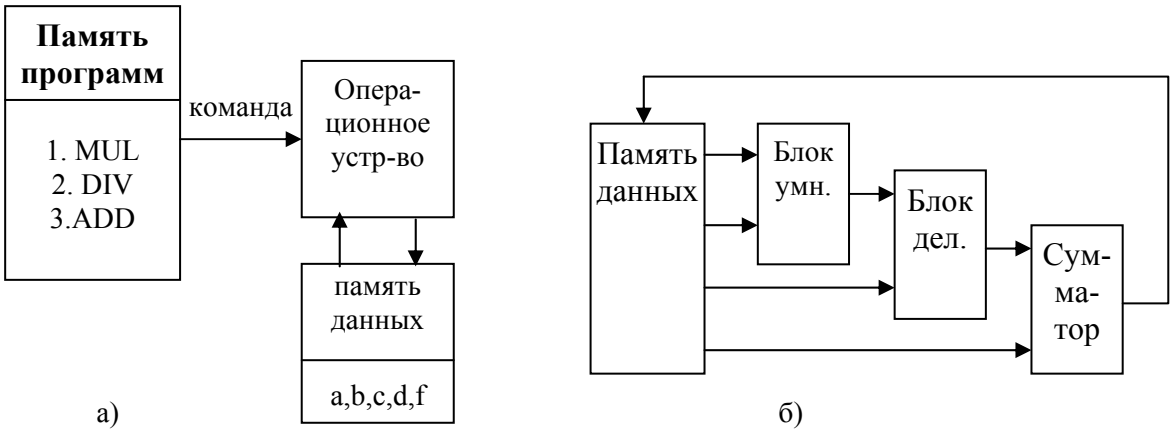
При программной реализации алгоритма вычислительный процесс организуется как последовательность процедур, выполняемых поочерёдно во времени на одном операционном устройстве (ОУ). Такое процедурное представление алгоритма удобно оформлять в виде блок-схемы алгоритма. При аппаратурной реализации алгоритма вычислительный процесс разворачивается в пространстве операционных блоков, соединённых между собой в соответствии с потоковым графом алгоритма и работающих параллельно во времени.

На рис. 1.1 показаны два варианта представления алгоритма, а на рис. 1.2 – два варианта его реализации. Очевидно, что во втором случае отпадает необходимость в программной памяти, так как программа вычислений заменяется схемой соединений операционных блоков.



а) б)

Рис. 1. 1. Процедурное (а) и потоковое (б) представление алгоритма вычисления значения f = ab/c + d



а) б)

Рис. 1. 2. Программная (а) и аппаратурная (б) реализации алгоритма f = ab/c + d

Сравнивая рассмотренные варианты, можно отметить, что при программной реализации сложность алгоритма влияет на длину программы, а при аппаратурной реализации – на количество оборудования. Время вычисления одного результата в обоих случаях может быть одинаковым (без учёта времени обращения к памяти). Однако аппаратурная реализация позволяет организовать конвейерную обработку, что существенно повысит её производительность. Это обстоятельство и определяет современные тенденции развития вычислительной техники, учитывая блестящие успехи современной микроэлектроники.

Очень быстрый рост степени интеграции современных микросхем, когда сверхбольшие интегральные схемы (СБИС) могут содержать сотни миллионов транзисторов в одном корпусе, диктует необходимость проектирования мощных

аппаратурных средств реализации алгоритмов. Однако разработанные алгоритмы решения прикладных задач, в том числе и цифровой обработки сигналов, мало ориентированы на аппаратурную реализацию. Необходимы специальные – аппаратурно-ориентированные алгоритмы, синтезированные с учётом требований технологии СБИС. В идеальном случае процесс разработки алгоритма должен быть совмещён с проектированием СБИС, так как топология СБИС изоморфна потоковому графу аппаратурно-реализуемого алгоритма.

Рассмотрим процесс решения задачи на универсальной ЭВМ, включающий ряд этапов, показанных на рис. 1. 3.

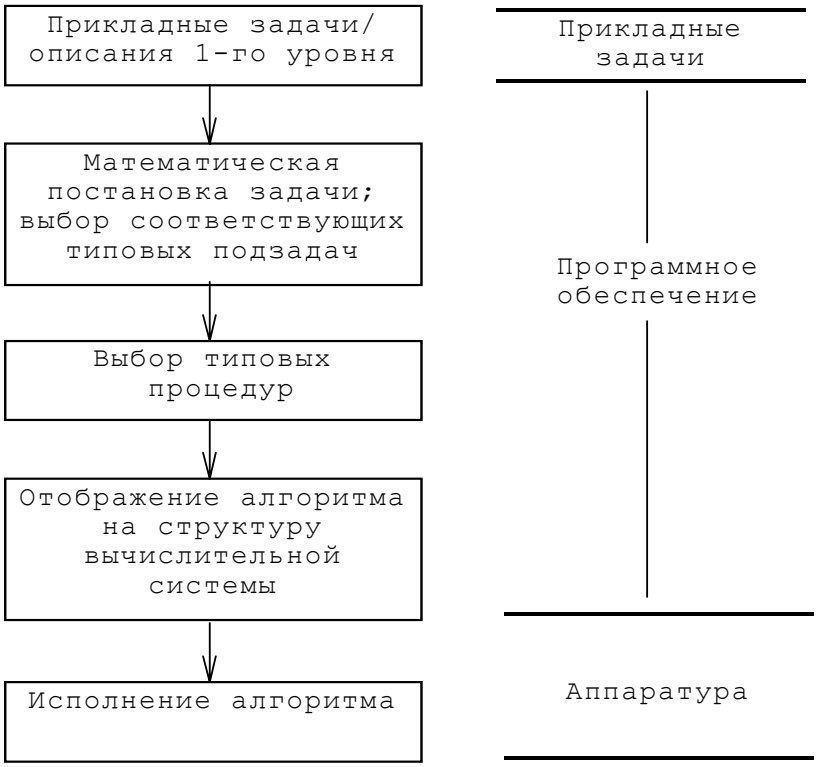


Рис. 1.3

На начальном этапе задача, возникающая в некоторой прикладной области, формулируется на естественном языке (составляется описание задачи). Затем осуществляется математическая постановка задачи и выбор соответствующих типовых подзадач. Следующим этапом является выбор типовых вычислительных процедур для реализации необходимых подзадач и отображение алгоритма на структуру вычислительной системы. После этого алгоритм исполняется на имеющейся вычислительной структуре.

В универсальных ЭВМ арифметико-логическое устройство (АЛУ) строится, как правило, на основе универсального сумматора. Такое АЛУ выполняет лишь элементарные операции типа сложения, сдвига и некоторые другие. Поэтому для исполнения алгоритма на таком АЛУ необходима программа, состоящая из таких операций.

Таким образом, выбор типовых подзадач, типовых процедур и отображение алгоритма на структуру вычислительной системы осуществляется в универсальной ЭВМ на этапе процедурного программирования.

Время решения задачи на такой ЭВМ прямо зависит от длины программы. Чем крупнее будет математическая функция, выполняемая аппаратурно, тем меньше будет длина программы, меньше обращений к памяти, а, следовательно, меньше время решения и выше производительность ЭВМ. По мере роста возможностей интегральной электроники увеличивается сложность задач, для которых возможна аппаратная реализация их решения, и граница между

задачами, реализуемыми аппаратурно и программно, сдвигается. То есть, аппаратурно реализуются уже типовые вычислительные процедуры, крупные математические и даже прикладные алгоритмы. С другой стороны, рост производительности процессоров позволяет решать более сложные задачи, традиционно решавшиеся аппаратным способом, программными средствами (примером может служить появление т.н. Winmodem'ов). Таким образом, граница между программными и аппаратными средствами при реализации алгоритмов постоянно перемещается.

1.2. Понятие об архитектуре и структуре ЭВМ

Аппаратные средства вычислительной техники (ВТ) строятся по иерархической схеме (от низших уровней к высшим):

1. Элементарные логические схемы (базовые вентили), в свою очередь построенные на нескольких интегральных транзисторах.
2. Типовые схемотехнические узлы - комбинационные схемы (триггеры, регистры, дешифраторы, одноразрядные и параллельные сумматоры).
3. Функциональные узлы ЭВМ, состоящие из нескольких типовых схем (блок регистров, запоминающее устройство, матричный умножитель, АЛУ и т.д.)
4. Подсистемы ЭВМ (процессор, числовой сопроцессор, контроллер внешнего устройства, система памяти, подсистема ввода-вывода и т.д.)
5. Автономные вычислительные машины.
6. Вычислительные системы, комплексы и сети.

Для описания сложной системы, которую представляет собой вычислительная машина, часто используют 2 понятия: **архитектура** и **структура**. Под архитектурой понимают абстрактное представление о ВМ с точки зрения программиста. Сюда входит описание программных средств, аппаратных средств и принципов организации вычислительного процесса на аппаратных средствах с помощью программных средств. В отличие от архитектуры структуру ВМ можно определить как совокупность аппаратных средств с указанием основных связей между ними.

Архитектура в более широком смысле определяется как концепция взаимосвязи элементов сложной структуры и включает в себя компоненты логической, физической и программной организации ВМ. В более узком смысле архитектура ВМ – это описание ее системы команд для программиста (архитектура системы команд).

Развернутое описание архитектуры должно включать:

- форму представления программ в ВМ и правила их интерпретации;
- основные форматы представления данных;
- способы адресации данных в программе;
- состав аппаратных средств ВМ и их характеристики;
- соотношение и взаимодействие аппаратных и программных средств.

К классическим архитектурам можно отнести, прежде всего, архитектуру фон-Неймана. Вот некоторые из характерных особенностей этой архитектуры:

1. Программное управление вычислительным процессом путем автоматического извлечения команд программы из памяти и **последовательного** их выполнения.
2. **Общая память** для программ и данных.
3. Одинаковое кодирование программ и данных.
4. Использование двоичной системы счисления.
5. Арифметическое устройство на базе двоичного сумматора.
6. Иерархическое построение памяти и др.

Вычислительная машина фон-Неймана на самом общем уровне имеет структуру, представленную на рис. 1.4. Она состоит из центрального процессора (ЦП), запоминающего устройства, образующих процессорное ядро ВМ, и внешних устройств (ВУ), взаимодействующих с ядром через систему ввода-вывода (СВВ). Сам процессор включает устройство управления (УУ), последовательно извлекающее из общей памяти программ/данных команды и управляющее их исполнением, и арифметико-логическое устройство (АЛУ) на базе двоичного сумматора, непосредственно исполняющее последовательности простых арифметических и логических операций под управлением УУ.

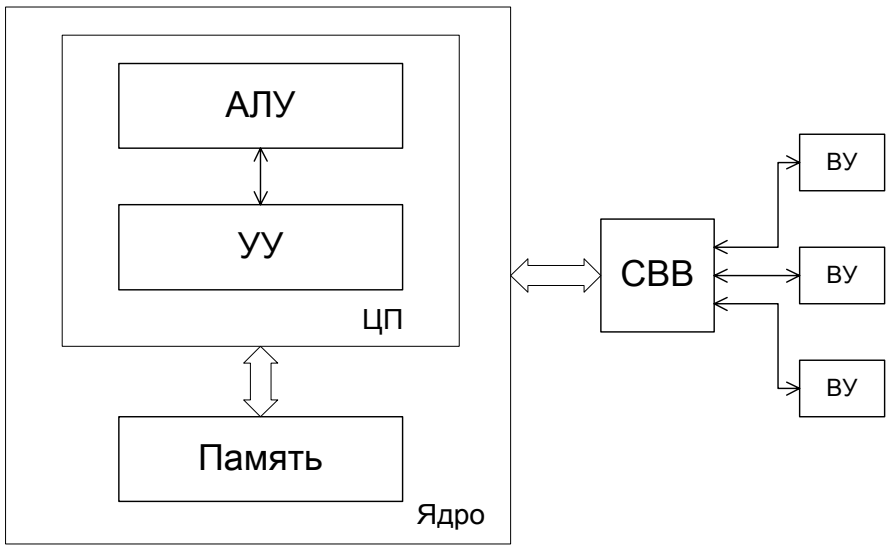


Рис. 1.4

Классической архитектурой примерно с середины 70-х годов можно считать и гарвардскую архитектуру. Основным ее отличием от архитектуры фон-Неймана является раздельная память программ и данных. Такая архитектура характерна для управляющих, встраиваемых, специализированных машин, программное обеспечение которых зачастую не меняется с момента изготовления («зашитые» программы), в то время как программа в машине фон-Неймана может даже менять саму себя в процессе работы.

Все остальное многообразие архитектур ВМ можно также отнести к не-фон-неймановским. Многие из них также уже являются почти классическими (как, например, векторные машины). Как правило, основным отличием не-фон-неймановских архитектур является распараллеливание работы, допускаемое ими. Современные ВМ довольно редко полностью соответствуют требованиям классической архитектуры фон-Неймана, тем не менее, одни из них в целом близки к ней, другие – сильно отличаются от «классической» вычислительной машины.

1.3. Характеристики вычислительных машин

Характеристики ВМ можно разбить на несколько групп. Прежде всего, в силу основного назначения ВМ – выполнять вычисления – наибольший интерес для пользователей представляет группа временных характеристик.

1. Временные характеристики. Важнейшими из них являются производительность и быстродействие ВМ. Часто эти две характеристики отождествляют, но мы будем их различать, отмечая архитектурные особенности построения различных ВМ.

Под *производительностью* (productivity, performance, throughput) понимают количество операций (задач), выполняемых (решаемых) на данной ВМ в единицу времени.

Быстродействие (speed, velocity) – величина, обратная времени выполнения одной операции (или задачи).

Различие между этими двумя определениями можно проиллюстрировать на следующих примерах. На рис. 1.5 представлены различные варианты построения ВМ. На рис. 1.5а ВМ включает единственный процессор, выполняющий обработку одного потока данных. Для такой системы с последовательной обработкой. производительность P равна быстродействию V.

На рис. 1.5б представлена система из N параллельно работающих процессоров, каждый из которых выполняет обработку своего пакета данных (своей задачи). Каждая задача обрабатывается за то же время, что и в случае единственного процессора (рис. 1.5а), то есть быстродействие параллельной системы не меняется (Vп = V).

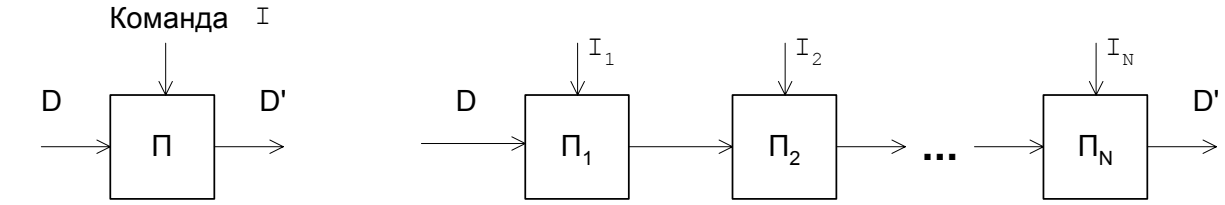
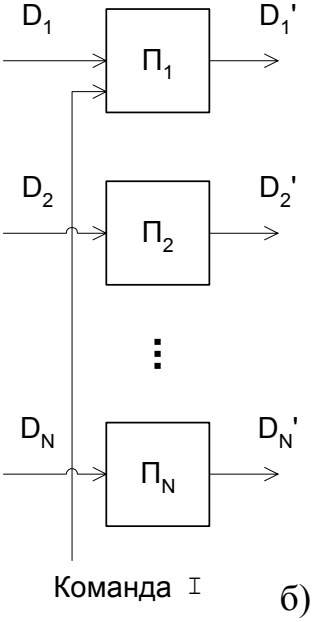


Рис. 1.5



В то же время, производительность параллельной системы в N раз выше, чем у одного процессора – Рп = N·P. На рис. 1.3в представлен вариант *конвейерной* обработки, при которой задача разбивается на N ступеней, каждая из которых выполняется на отдельном оборудовании. Время выполнения задачи в целом Тк = N·t, где t –время работы одного процессора. Тогда быстродействие системы Vк = 1/(N·t) = V/N, где V – быстродействие одного процессора. В то же время, производительность системы может быть выше, так как на разных ступенях конвейера в одно и то же время могут параллельно выполняться разные задачи на разных этапах.

В самом лучшем случае Рк = 1/t, то есть Рк = Р, где Р – производительность одного процессора.

Производительность является в общих случаях более важной прикладной характеристикой ВМ. В большей степени производительность – это архитектурная характеристика. Быстродействие в большей степени является технологической характеристикой ВМ. В некоторых случаях именно быстродействие машины оказывается более важной характеристикой, например, когда речь идет о скорости отклика системы на какое-либо входное воздействие (примером может служить ВМ для управления каким-либо производственным процессом).

Для измерения производительности служат различные тесты, так называемые тестовые смеси, в которых в определенном процентном соотношении присутствуют операции различной сложности, наборы типовых алгоритмов решения прикладных задач различного назначения (например, решение систем уравнений, архивирование файлов, трассировка печатных плат) и т.д. Выражают производительность и быстродействие либо в конкретных задачах (например, в количестве решенных систем уравнений в секунду), либо – в относительных интегральных показателях, полученных по результатам тестирования, по отношению к известным базовым моделям ВМ (например, SPEC, iComp), либо – в количестве операций в секунду. В последнем случае используют такие единицы, как OPS – операций в секунду, IPS – инструкций в секунду, FLOPS – операций с плавающей запятой в секунду. IPS и OPS являются очень субъективными единицами, поскольку никак не раскрывают сложности операций, в которых выражена производительность, если это специально не оговорено. Показатель производительности в FLOPS более объективен, так как операции более четко определены и, кроме того, различные операции с плавающей запятой (сложение, умножение) более однородны по сложности, чем, например, те же операции с фиксированной запятой.

2. Разрядность ВМ (ширина разрядной сетки). Разрядность также является важной прикладной характеристикой ВМ. Разрядность влияет на точность обработки и на производительность. Под разрядностью ВМ понимают обычно разрядность обрабатывающего устройства (АЛУ) процессора, иногда - разрядность шины данных процессора. Чем больше разрядность ВМ, тем с большей скоростью она может обрабатывать многоразрядные числа. Конечно, повысить точность обработки можно с помощью программных средств, однако при небольшой ширине разрядной сетки ВМ это приведет к падению производительности.

3. Используемая система счисления. Чаще всего в ВМ по понятным соображениям используется двоичная система счисления. Тем не менее, наряду с двоичной системой иногда применяют и другие системы, например, двоично-десятичное кодирование, троичную систему и другие. В специализированных ВМ находит применение непозиционная система счисления в остаточных классах (СОК), особенностью которой является отсутствие межразрядных переносов при сложении. В системах с последовательной обработкой разрядов применяют

троичную знакоразрядную систему счисления, которая позволяет проводить обработку чисел старшими разрядами вперед.

4. Характеристики и емкость системы памяти. Широко известно утверждение, бытующее в популярной литературе по ВТ (например, в компьютерных журналах), что память (прежде всего – оперативная) помимо процессора является важнейшим ресурсом ВМ. В значительной степени это утверждение следует признать верным. Производительность (или – пропускная способность) памяти довольно часто является узким местом, ограничивающим производительность ядра ВМ и ВС в целом, поэтому важной характеристикой системы памяти является не только ее объем, но и производительность.

5. Скорость обмена между компонентами ВМ. Так же, как память может ограничивать производительность ядра ВМ, так же и система ввода-вывода (СВВ) может ограничивать производительность ВМ в целом за счет своей низкой пропускной способности. Для повышения пропускной способности СВВ необходимо применять современные быстродействующие интерфейсы, согласующие буферы данных, кэширование данных и так далее.

6. Характеристики надежности и другие эксплуатационные характеристики.

Надежность - свойство системы выполнять заданные функции, сохраняя значение заданных показателей в установленных пределах в течении заданного промежутка времени.

Надежность характеризуется рядом показателей.

Безотказность. Под *отказом* понимают нарушение работоспособности ВМ, для устранения которого требуется вмешательство человека. Безотказность характеризуется временем наработки на отказ То, то есть временем, в течение которого гарантированно не будет отказа.

Долговечность – работоспособность системы до наступления предельного состояния, при котором дальнейшая эксплуатация системы нецелесообразна.

Ремонтопригодность – приспособленность вычислительной машины к обнаружению отказов. Характеризуется временем восстановления работоспособности при отказе Тво.

Достоверность функционирования ВМ – безошибочность проводимых в ВМ преобразований информации. Характеризуется частотой возникновения сбоев. Под *сбоем* понимают временное нарушение работоспособности ВМ, восстановление после которого происходит без вмешательства извне, то есть средствами самой системы. Характеризуется средним временем наработки на сбой – Тс.

Надежность в целом характеризуется интенсивностью (вероятностью) отказов.

Важной эксплуатационной характеристикой ВМ является *коэффициент использования*:

$Ки = To / (Tво + To + tпо)$,

где tпо - время профилактического обслуживания, приходящееся на один отказ.

Другим широко используемым показателем является *коэффициент готовности* Кг, он определяет вероятность нахождения ВМ в работоспособном состоянии в период между профилактическими работами:

Ки = То / (Тво + То).

Системы с высоким показателем коэффициента готовности называют *системами высокой готовности*.

7. Показатели эффективности ВМ. Обычно эти показатели связывают с отношением «производительность – стоимость», либо – «производительность – сложность» (для интегральных устройств – «производительность – количество транзисторов») :

Э = Р/С.

Для конечного пользователя более важным является отношение «производительность/стоимость», в то время как для разработчиков и исследователей ВТ часто интереснее показатель «производительность – сложность», поскольку стоимость помимо сложности зависит от массовости производства и других факторов, имеющих скорее экономический, чем технический характер.

1.4. Общая классификация вычислительных машин

Вычислительные машины можно классифицировать по разным признакам, в том числе: по производительности, назначению, элементной базе и т.д. К основным из них можно отнести, например, следующие:

1. *По способу представления информации :*
- ВМ непрерывного действия (аналоговые ВМ);
- ВМ дискретного действия (цифровые ВМ) ;
- гибридные ВМ (смешанного типа).
2. *По назначению (степени специализации):*
- ВМ общего назначения ;
- специализированные и проблемно-ориентированные ВМ.
3. *По физическому эффекту, используемому для представления, кодирования и обработки информации:* электронные ВМ; магнитные ВМ; механические ВМ; электромеханические; криогенные ВМ; оптические ВМ; пневматические ВМ; гидравлические ВМ и др.
4. *По количеству вычислительных устройств и степени распределенности:*
- автономные ВМ ;
- вычислительные системы ;
- вычислительные комплексы ;
- вычислительные сети.

Вычислительная система - сложная совокупность аппаратных средств, в том числе двух и более процессоров, соединенных внутренними шинами и реализующих общие программы вычислений. (Например, параллельные ЭВМ.)

Вычислительный комплекс - совокупность двух и более ЭВМ одного или различных типов, предназначенных для решения общего класса задач и соединенных между собой посредством общей (внешней) памяти (с косвенной связью) или через каналы ввода/вывода (с прямой связью). (Например, две ЭВМ, подсоединенные к одному массиву дисков.)

Вычислительная сеть - множество ЭВМ, соединенных стандартными телекоммуникационными каналами связи или стандартными каналами передачи данных (например, ЛВС).

5. *По производительности* можно достаточно условно разделить все ВМ на суперкомпьютеры (ВМ наибольшей производительности на данный момент), большие ВМ (супермини-ВМ), мэйнфреймы, то есть мини-ВМ, высокопроизводительные рабочие станции, микро-ВМ, в том числе – автономные микро-ВМ и встраиваемые ВМ различной производительности, которая может быть сравнительно небольшой. Деление это весьма условно, и границы между машинами одного и другого класса размыты, кроме того, по мере роста общей производительности средств ВТ количественные показатели, отмечающие эти границы, естественно, меняются. Так, на момент написания этих строк, подобные границы можно было бы обозначить следующим образом – к суперкомпьютерам можно отнести ВМ с производительностью от нескольких сотен гигафлопс до десятяков терафлопс, к супермини – до сотен гигафлопс, к мейнфреймам – до нескольких десятков гигафлопс, а производительность микро-ЭВМ уже может достигать нескольких гигафлопс. То есть в настоящее время границы между промежуточными группами ВМ стираются, наблюдается тенденция к миниатюризации высокопроизводительных вычислителей и можно говорить об уменьшении числа таких промежуточных групп с выделением в отдельные группы только супер-ЭВМ, мейнфреймов и общедоступных настольных или миниатюрных микро-ЭВМ.

6. *По сфере применения* можно выделить ВМ, предназначенные для выполнения научных и инженерных расчетов, управляющие и индустриальные ВМ, встраиваемые ВМ, ВМ, специализированные для обработки сигналов, персональные ВМ и др.

7. Важной характеристикой, непосредственной связанной с определением архитектуры машины, является *количество процессоров в ВМ.* Соответственно можно выделить однопроцессорные и многопроцессорные ВМ.

8. *По способу управления*. Наряду с традиционными ВМ, управляемыми потоком инструкций (команд), выделился достаточно обширный класс ВМ, управляемых потоком данных (потоковые архитектуры). По крайней мере элементы потоковых архитектур используются во многих современных суперскалярных микропроцессорах.

Классификация не ограничивается приведенными признаками, мы привели только некоторые из возможных, так как многообразие средств ВТ достаточно велико.

1.5. Основные пути повышения производительности ЦВМ

Одной из наиболее важных прикладных характеристик ВМ является ее производительность. Повышение производительности – зачастую главное требование, стоящее перед разработчиками ВТ. Можно выделить несколько основных путей решения этой задачи:

1 Совершенствование технологии производства ЭВМ («физический» путь) - повышение быстродействия логических элементов.

2. Распараллеливание вычислений.

3. Конвейеризация вычислений.

4. Специализация вычислений.

5. Аппаратная реализация сложных функций.

Совершенствование технологии в основном направлено на уменьшение геометрических размеров, снижение потребляемой мощности и уменьшение времени переключения логических вентилей. (В настоящее время - геометрические размеры порядка 0.1 мкм, время переключения - порядка 0.1нс). Но - этот путь ограничен физическими пределами.

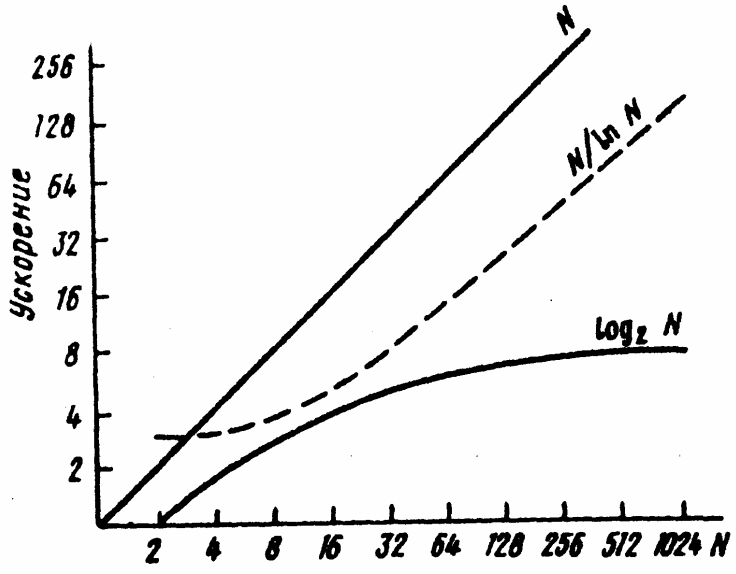


Рис. 1.6

Альтернативные пути - архитектурные (логические), и прежде всего - параллельные и конвейерные вычисления.

Параллелизм можно определить как возможность одновременного выполнения разных математических или служебных операций. *Распараллеливание* - нахождение алгоритма решения задачи, использующего параллелизм и реализация этого алгоритма в ВС.

Параллельные ВС - это многопроцессорные ВС, в которых параллелизм используется для повышения производительности при решении задач за счет одновременного выполнения разных операций на разных процессорах или обрабатывающих устройствах одного процессора..

В идеальном случае для ВС из N процессоров производительность : $P_N = P_0N$, где P_0 - производительность одного процессора. Однако на практике ситуация зачастую оказывается иной.

В литературе по ВС часто приводят зависимости ускорения в ВС от числа процессоров в системе, приведенные на рис. 1.6 Наиболее оптимистичная оценка соответствует ожиданиям (ускорение в N раз), но встречается только для некоторых ВС (например, систолического типа).

Более пессимистические оценки Амдаля (N/(ln N)) и Минского ($\log_2$N) объясняются рядом обстоятельств.

В частности, можно выделить три проблемы распараллеливания:

1. Распараллеливание алгоритма (математическая проблема).
2. Распараллеливание вычислительной структуры (архитектурная и схемотехническая, системная проблема)
3. Перенос алгоритма на структуру.

Эти проблемы возникают на разных этапах построения параллельной ВС. Первая проблема связана со спецификой решаемой задачи и выбранного метода решения. Некоторые задачи очень хорошо распараллеливаются (например, многие задачи обработки матриц), некоторые - очень плохо. Вторая - с решением вопросов взаимодействия параллельных процессоров (например, обмен с памятью, межпроцессорный обмен, синхронизация и др.) Третья - с возможностью реализации выбранного распаралеленного алгоритма на данной параллельной структуре.

Конвейеризация вычислений - разбиение вычислений на последовательные этапы с целью реализации этих этапов на отдельных ступенях конвейера для повышения производительности.

Конвейер - устройство, состоящее из N последовательно соединенных частей (ступеней конвейера), каждая из которых выполняет очередной шаг вычислений за время t (такт конвейера). Таким образом, для решения задачи, требующей N шагов, потребуется время, равное $t\,N$, однако производительность конвейера может быть достаточно велика, поскольку освобождающиеся ступени могут заполняться новыми данными. В результате на каждом такте конвейер может выдавать очередной результат.

Время решения одной задачи на конвейере:

$$T_{\text{реш}} = N\,t.$$

Производительность конвейера:

$$P = N'/(T_0 + N\,t);$$

где N' - количество задач, поступающих на вход конвейера.

T_0 - время подготовки данных.

Из последней формулы видно, что при $T_0 << N\ t$ и $N' \rightarrow N$ производительность конвейера стремится к величине $1\ /\ t$ (пиковая производительность конвейера). В то же время, при T_0 сопоставимом с $N\ t$ (большое время загрузки и подготовки конвейера), или при малом N' производительность конвейера будет ниже.

Так же, как и в случае распараллеливания, при конвейеризации возникают проблемы создания алгоритма, создания структуры и перенесения алгоритма на структуру. При этом к алгоритму предъявляют требования:

отсутствие (минимальное количество) циклов и развилок ;

возможность разбиения на шаги одинаковой длительности (и сложности);

последовательное и равномерное перемещение данных по алгоритму и др.

Часто конвейерный режим используют в векторных ВС. Векторные ЭВМ и ВС выполняют не только скалярные операции (над числами), но и операции над векторами (массивами чисел), при этом векторные операции выполняются параллельно для всех элементов вектора, либо - на конвейере. В последнем случае производительность повышается за счет одновременного выполнения на конвейере нескольких векторных операций :

Ускорение = Тск (m-1) Q / (mt + Тп),

где Тск - время скалярной операции, m- длина вектора, t - такт конвейера, Q - количество одновременно выполняемых векторных операций на конвейере (количество задач), Тп - время подготовки конвейера (заполнение конвейера).

Специализация вычислений позволяет повысить производительность за счет аппаратной реализации решения задачи в целом, либо какой-либо сложной части этой задачи, то есть методом программирования аппаратной структуры. Выигрыш в производительности достигается за счет сокращения расходов на управление, а также за счет снижения универсальности устройства.

В то же время, *аппаратная реализация сложных функций* (арифметических, матричных операций и т.д.) может применяться и в универсальных ВС без снижения функциональности, за счет только увеличения стоимости устройств и придания им дополнительных функций.

В дальнейшем мы увидим примеры применения этих подходов в различных ВС.

2. ОРГАНИЗАЦИЯ СИСТЕМ ПАМЯТИ

2.1. Характеристики и классификация запоминающих устройств. Иерархия систем памяти

Под *запоминающими устройствами* (ЗУ, память) будем понимать совокупность устройств для запоминания, хранения и выдачи информации. Память является одним из основных ресурсов компьютера, влияющим как на производительность, так и на функциональность вычислительной машины.

К основным характеристикам устройств памяти можно отнести:

1) Временные характеристики :

- *быстродействие* - определяется временем выборки, временем обращения и другими параметрами. Время обращения складывается из различных составляющих, например:

$$t_{обрЧТ} = t_{дост} + t_{чт} + t_{рег},$$

где $t_{обрчт}$ - время обращения при чтении, $t_{дост}$ – время доступа к данным, $t_{рег}$ – время регенерации (для динамической памяти), $t_{чт}$ – время собственно чтения;

$$t_{обрЗП} = t_{дост} + t_{подг} + t_{зп},$$

где $t_{обрЗП}$ - время обращения при записи, $t_{подг}$ – время подготовки данных, $t_{зп}$ – время собственно записи. Таким образом, процесс чтения/записи ЗУ в общем случае включает ряд этапов разной сложности и длительности.

- *производительность* – определяется пропускной способностью ЗУ, то есть – объемом информации, который можно считать/записать из/в ЗУ в единицу времени. Для оценки производительности часто используют показатель длительности цикла обращения к памяти $t_ц$, под которым понимают минимальное время между сменой информации на выходе/ входе ЗУ. Длительность цикла не всегда совпадает с временем обращения, в частности, при конвейеризации ЗУ можно добиться увеличения производительности при достаточно большой величине $t_{обр}$ за счет разделения общей задачи чтения/записи на последовательные ступени конвейера.

2) Важнейшей потребительской характеристикой ЗУ является его *объем*, или *емкость памяти (E),* то есть количество запоминаемой информации. В зависимости от типа ЗУ, его места в вычислительной системе, объем может меняться от десятков байт (для регистровой памяти ЦП) до десятков и сотен гигабайт (для массивов накопителей на магнитных дисках).

Наряду с характеристикой емкости памяти применяют также удельную емкость по отношению к единице площади или объема кристалла :

$$E_{уд} = E/S_{кр}.$$

Такая характеристика в большей степени характеризует технологические особенности ЗУ.

3) Третьей важнейшей потребительской характеристикой ЗУ, как и любого вычислительного устройства, является его стоимость, которая также может меняться в самых широких пределах в зависимости от объема, производительности и других характеристик. Распространенной характеристикой является удельная стоимость в расчете на единицу информации (стоимость одного бита/байта, кило- и мегабайта и т.д.)

Помимо перечисленных можно отметить множество других характеристик ЗУ, в том числе: технологию изготовления, потребность во внешнем источнике питания для хранения информации, длительность хранения, количество циклов чтения и записи, геометрические размеры, и так далее.

С учетом приведенных характеристик, а также – назначения ЗУ, места, занимаемого ЗУ в вычислительной системе, можно привести, например, следующую классификацию ЗУ:

1. По удаленности от процессора :
- сверхоперативная (регистры процессора, КЭШ память);
- основная (оперативная) память ;
- дополнительная память (внешняя) ;
- вторичная память (также внешняя) ;
- массовая память (внешняя, как правило, на доступных сменных носителях).

2. По организации записи :
- постоянное запоминающее устройство – ПЗУ (ROM – read-only memory) – однократно программируемое изготовителем устройство только для чтения;
- перепрограммируемое запоминающее устройство – ППЗУ (PROM) – возможно перепрограммирование, которое, однако, требует специальной процедуры, кол-во циклов записи намного меньше циклов чтения;
- оперативное запоминающее устройство - ОЗУ (RAM – random access memory) - количество циклов чтения может совпадать с количеством циклов записи.

Строго говоря, приведенные отечественные и импортные сокращения для двух основных типов памяти не вполне точно отражают приведенное деление памяти по организации записи, но являются исторически сложившимися и общепринятыми.

3. По организации доступа :
- с последовательным доступом ($t_{дост}$ меняется для различных адресов или участков памяти – чем старше адрес, тем больше время доступа);
- с прямым доступом ($t_{дост}$ = const для различных адресов или участков памяти).

4. По организации поиска ячеек в памяти:
- «М-поиск» – поиск по месту (например, в адресном ОЗУ);
- «В-поиск» – поиск по времени (например, при работе с накопителем на магнитной ленте).

5. По физическому эффекту (технологии), используемому для запоминания и хранения информации :
- полупроводниковая память;
- магнитная;
- магнитооптическая;
- оптическая;
- электростатическая и др.

6. ОЗУ по способу хранения делится на :
- статическое (на триггерах);
- динамическое (на конденсаторах).

7. По способу адресации:
- адресная память;
- стековая память;
- ассоциативная память.

8. По организации памяти в систему:
- память с расслоением;
- виртуальная память;
- кэш-память;
- различные варианты блочно-конвейерных систем.

9. По зависимости от источника питания:
- энергозависимая;
- энергонезависимая.

Как и ранее, при классификации вычислительных машин, отметим, что выбранные классификационные признаки не являются всеобъемлющими или обязательными, просто они отражают некоторые важные особенности классифицируемых систем.

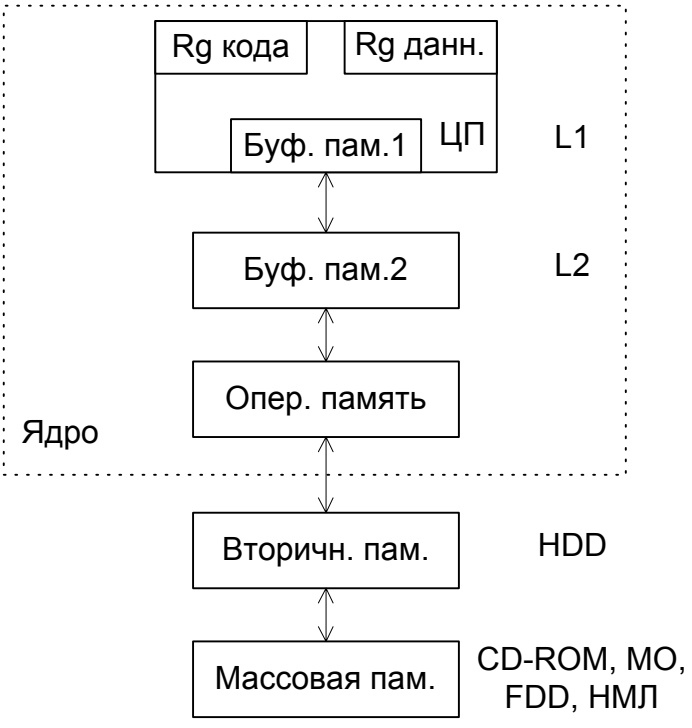


Рис. 2.1

Рассматривая характеристики и классификацию ЗУ, с учетом их многообразия нельзя не упомянуть об иерархии систем памяти в составе вычислительной системы. Как мы помним, принцип иерархического построения систем памяти заложен еще в фон-неймановской архитектуре, в те годы, когда большинства современных ЗУ и их типов не существовало. Однако и тогда существовала относительно быстрая и дорогая энергозависимая оперативная память, и внешняя память – более дешевая, намного более медленная, но при этом энергонезависимая. Сейчас

иерархия выглядит намного сложнее, но общий принцип ее построения остается в основном неизменным (Рис.2.1).

На верхнем уровне иерархии располагается наиболее быстрая и дорогая регистровая память процессора, а также – буферная кэш-память первого уровня, расположенная в кристалле процессора. К ней примыкает кэш-память второго уровня, выполняемая в одном корпусе с процессором, либо – на системной плате. На следующем уровне находится оперативная (чаще всего – динамическая) память достаточно большого объема. Эти уровни вместе с процессорами образуют ядро ВС в архитектуре фон-Неймана. На более низких уровнях располагается внешняя память – внешние устройства, взаимодействующие с ядром по каналам ввода-вывода. В качестве вторичной памяти можно указать НЖМД (HDD) – накопители на жестких магнитных дисках – пожалуй, наиболее быстродействующую внешнюю память, при этом со значительным объемом. К массовой памяти можно отнести разнообразные сменные носители информации, различающиеся как по объему, так и по времени доступа (накопители на гибких магнитных дисках, магнитной ленте, CD-ROM – диски и т.д.), которые объединяет, пожалуй, относительно низкая удельная стоимость.

Легко заметить, что при движении по иерархии сверху вниз происходит снижение удельной стоимости хранения информации, рост объемов ЗУ и - падение производительности.

Подобное построение систем памяти в ВС объясняется, с одной стороны, различной функциональной направленностью ЗУ (оперативное хранение небольших объемов информации в ОЗУ, либо – долговременное хранение больших объемов данных на дисковой памяти), а с другой - попыткой достичь более-менее приемлемого соотношения между ценой и производительностью (а также функциональностью) вычислительной системы, что являлось актуальным как на заре вычислительной техники, так и сейчас.

2.2. Организация адресной памяти

Отличительным признаком адресной памяти является организация доступа к ячейкам памяти по адресам, то есть – по номерам, которые поступают на вход ЗУ в закодированном виде, затем – декодируются тем или иным образом для выбора определенного запоминающего элемента (ЗЭ) или их группы. Подобная схема соответствует в большей степени устройствам с М-поиском, для которых время доступа является постоянной величиной, не зависящей от адреса.

Адресная память с М-поиском (под которой чаще всего подразумевают полупроводниковую память) на самом общем уровне включает в себя массив запоминающих элементов (триггеров, регистров, управляемых конденсаторов и т.д.), адресные дешифраторы для декодирования адреса ячейки в управляющие импульсы по шинам управления, усилители адресных и разрядных линий, а также все остальные необходимые логические схемы для осуществления выборки, считывания и записи и управления ЗУ.

Различные варианты организации памяти с М-поиском связаны, прежде всего, с различными способами построения массива ЗЭ и декодирования адреса. С этой точки зрения выделяют память типа 1D, 2D, 2,5D, 3D, 4D – по количеству измерений массива ЗЭ. В памяти типа 1D массив имеет 1 измерение, то есть адресуется каждый бит памяти. При достаточно большом объеме ЗУ это приводит к сложным схемам дешифраторов и огромному количеству служебных линий, трассировка которых внутри кристалла вызывает проблемы, а площадь, занимаемая ими, сопоставима с площадью массива самих ЗЭ. В 2D-памяти (рис. 2.2) адресуются не отдельные биты, а слова, что улучшает общую картину.

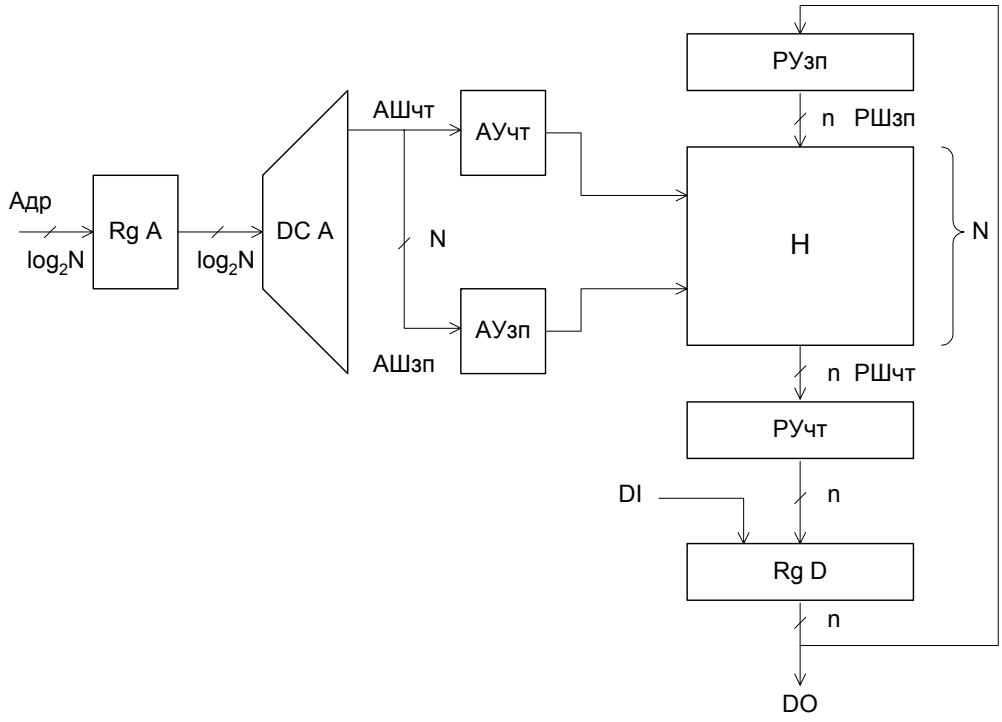


Рис. 2.2

Для большей экономии кристалла необходимо использовать слова еще большей разрядности, что входит в противоречие с разрядностью шин данных ВС, и создает дополнительные неудобства. Для их преодоления используют организацию типа 2,5D (рис. 2.3), при которой слова системной разрядности (16, 32, 64 и т.д.) объединяются в группы, адрес ячейки при декодировании в ЗУ делится на 2 части, большая из них используется для выбора группы, а меньшая – для выбора слова внутри группы.

При 3D организации (рис. 2.4) массив ЗЭ имеет два измерения, то есть выбор ячейки (слова) осуществляется по двум координатам, при этом адрес ячейки делится на две равные части, каждая из которых используется для выбора одной из линий по одной из двух координат. В результате и количество линий, и сложность адресных дешифраторов уменьшается. Дальнейшее развитие такого подхода приводит к памяти с организацией 4D и т.д.

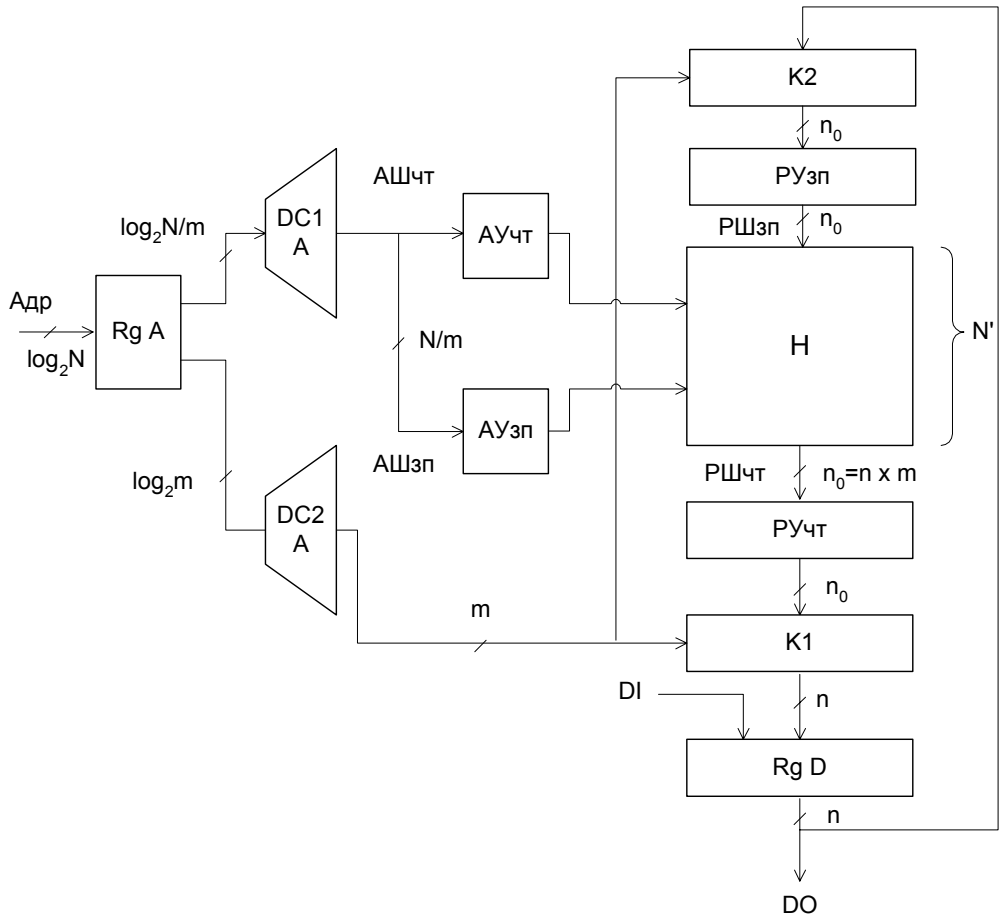


Рис. 2.3

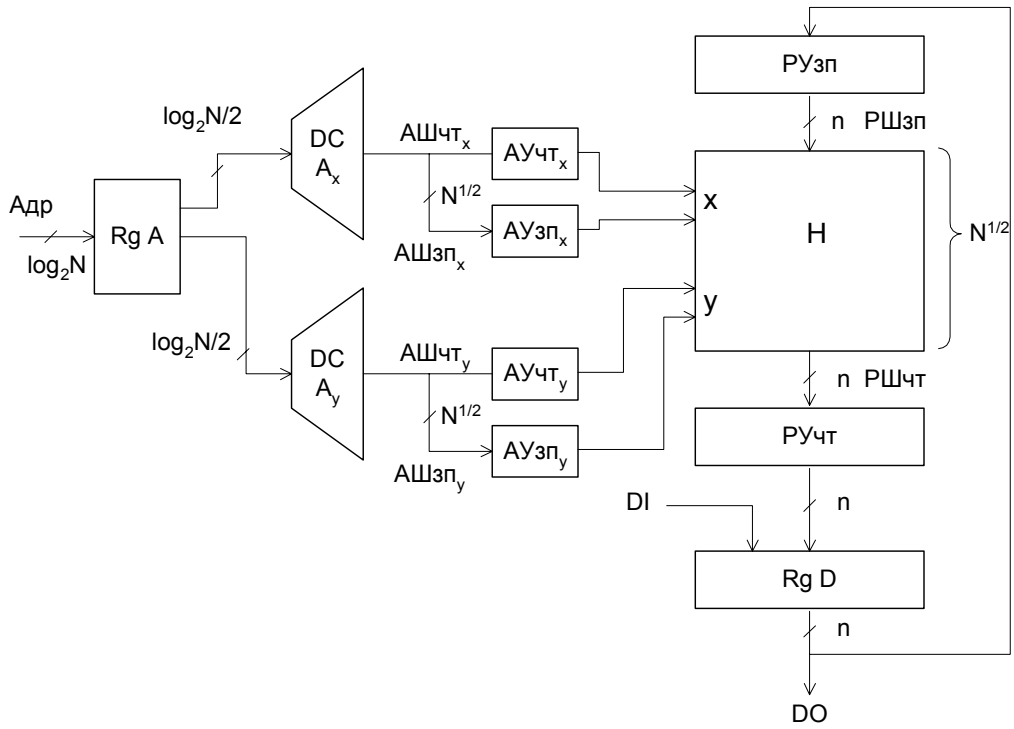


Рис. 2.4.

Память с В-поиском также использует адресный способ выборки ячейки (блока) информации, однако в таких устройствах адрес выступает не как

источник кода для дешифратора, а скорее – как инициализирующее значение для счетчика, который отсчитывает количество последовательно считанных, либо – просмотренных блоков. При обнулении счетчика последний блок записывается/считывается из массива накопителя. Физически к памяти такого типа можно отнести дисковую память (с определенной долей условности, если рассматривать подсистему «головка чтения/записи – дорожка»), а также, в более явном виде – накопители на магнитной ленте.

ЗУ с *В-поиском* в процессе подсчета блоков могут использовать внешнюю синхронизацию, либо – внутреннюю, то есть быть самосинхронизируемыми. В последнем случае синхронизация осуществляется с помощью адресных меток, которыми снабжен каждый блок данных, и которые подсчитываются устройством при выполнении последовательного доступа.

2.3. Безадресная стековая память

В стековой памяти (памяти магазинного типа, организованной по принципу «Последним вошел – первым вышел» - LIFO – “Last In - First Out”) все операции чтения и записи осуществляются относительно указателя стека (SP-stack pointer). Указатель стека указывает на ячейку памяти, содержащую последнее внесенное в стек слово. Стековая память может организовываться программно-аппаратным или аппаратным способом. Команды обращения к стеку не содержат адресной части, либо эта часть является относительной величиной, прибавляемой к указателю. Это позволяет сократить длину программы, так как нет необходимости указывать достаточно длинные адреса, а также – упростить схему ЗУ при аппаратной реализации стека.

В то же время при работе со стековой памятью приходится осуществлять фактически последовательный доступ, кроме того, может происходить т.н. переполнение стека – при попытке записать в полностью заполненный стек очередное значение, либо при считывании из пустого стека.

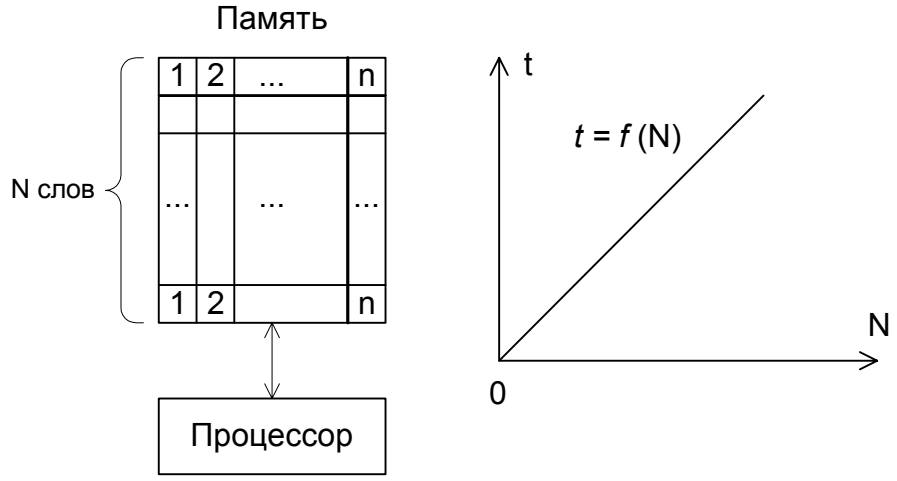
Использование стековой памяти будет более эффективным, если процессор, работающий со стеком, будет поддерживать специальные стековые команды - не только «занести в стек» и «считать из стека», но и такие, как - «сложить два числа на вершине стека», «переставить элементы стека» и т.д. Такие команды часто используются в RISC-процессорах, в микроконтроллерах, управляющих ЭВМ.

2.4. Ассоциативная память

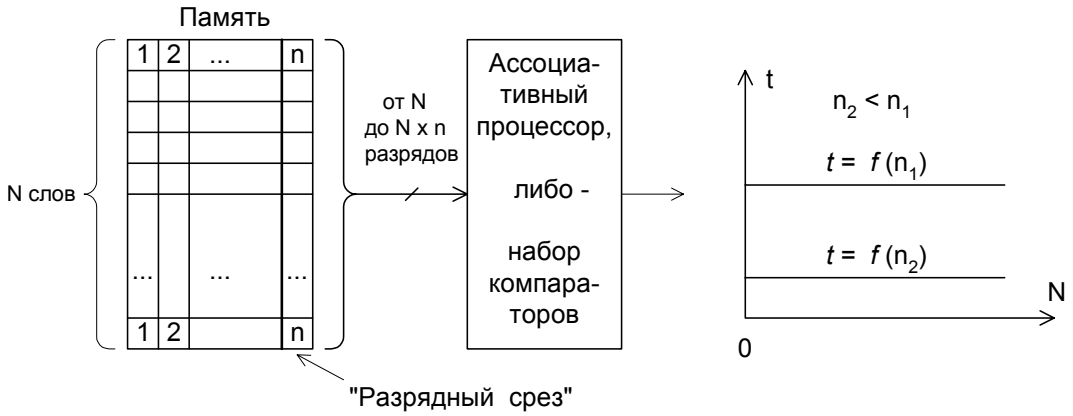
Под ассоциативной памятью (АП) подразумевают вариант организации памяти, при котором адресная информация, используемая для выборки слова из памяти, содержится в самих словах памяти. Чтение/запись осуществляется для тех слов, адресная часть которых (так называемый «тэг») полностью или частично совпадает с заданной.

Ассоциативная память может быть организована как программным, так и аппаратным путем. При программной реализации понятие АП используется в основном как модель взаимодействия программы (процессора) с источником данных. Например, в реляционных базах данных для ускорения поиска нужной информации широко используются т.н. ключевые поля, которые входят в состав каждой записи БД. Для быстрого поиска по ключам используют специальные индексные файлы, построенные, например, по принципу двоичных деревьев. Адресной информацией в данном случае является не номер записи, а содержимое, например, поля кода товара, или – фамилии человека. Индексные файлы же позволяют укорить процедуру поиска.

При аппаратной организации АП большую роль играют, во-первых, аппаратные средства поиска, различные быстродействующие компараторы (схемы сравнения), а во-вторых- вариант организации поиска. В частности, в АП часто используется принцип «вертикальной» обработки и разрядных срезов (рис. 2.5). При обычной «горизонтальной» обработке (рис. 2.5а) для отыскания нужного слова в массиве ячеек слова просматриваются последовательно, по адресам, то есть как бы горизонтально, если представить себе массив ячеек как вертикальный столбец.



а) «горизонтальная» обработка



б) «вертикальная» обработка

Рис. 2.5

При вертикальной обработке (рис. 2.5б) все слова просматриваются одновременно. При этом, если осуществлять сравнение искомого тэга со всеми разрядами всех тэгов слишком накладно, то используются вертикальные разрядные срезы (РС) всех слов накопителя. После первого сравнения отсекаются все слова, имеющие первый бит, несовпадающий с заданным тэгом, затем анализируется следующий РС и т.д.

Таким образом, отличительные особенности АП:

1. Операции в памяти выполняются не над определенной ячейкой. а относятся сразу к группе или ко всем элементам.
2. Основной операцией в АП является операция поиска или сравнения.
3. Время поиска в АП может не зависеть от числа ячеек в памяти.

При аппаратной организации АП выделяют 4 варианта :

1. Память с полным параллельным доступом (осуществляется параллельное сравнение всех тэгов с заданным по всем разрядам) – самый высокопроизводительный и самый дорогой вариант.
2. Память с последовательной обработкой разрядных срезов (РС). Время поиска (доступа) в такой памяти пропорционально разрядности тэгов.
3. Память с последовательной обработкой слов («горизонтальная обработка») - время поиска пропорционально числу слов в памяти. Фактически этот вариант только условно можно отнести к АП, и то в случае, когда сравнение каждого тэга с заданным осуществляется аппаратным способом.
4. Частично-ассоциативная память. Компромиссный вариант, в котором выделяются несколько групп слов (блоков слов), в каждой из которых производится последовательный поиск, но все группы обрабатываются параллельно, либо – наоборот, группы обрабатываются последовательно, а внутри группы ведется полностью ассоциативный поиск, или поиск по срезам.

На рис. 2.6 приведен пример структура блока ассоциативной памяти.
На рисунке использованы следующие обозначения :

RgАП – регистр адресного признака,

RgM – регистр маски,

RgD – регистр данных,

КС – комбинационная схема,

RgC – регистр совпадений,

ФС – формирователь сигналов (1 - нет совпадений; 2 - одно совпадение; 2 – более одного совпадения),

Н – накопитель;

N – количество слов в устройстве памяти (накопителе);

n – количество адресных разрядов в слове (тэге); n-ый разряд используется для указания занятости ячейки; m – разрядность собственно информационной части слова, не используемой для адресации.

Маска используется для выделения тех разрядов, которые должны участвовать в сравнении.

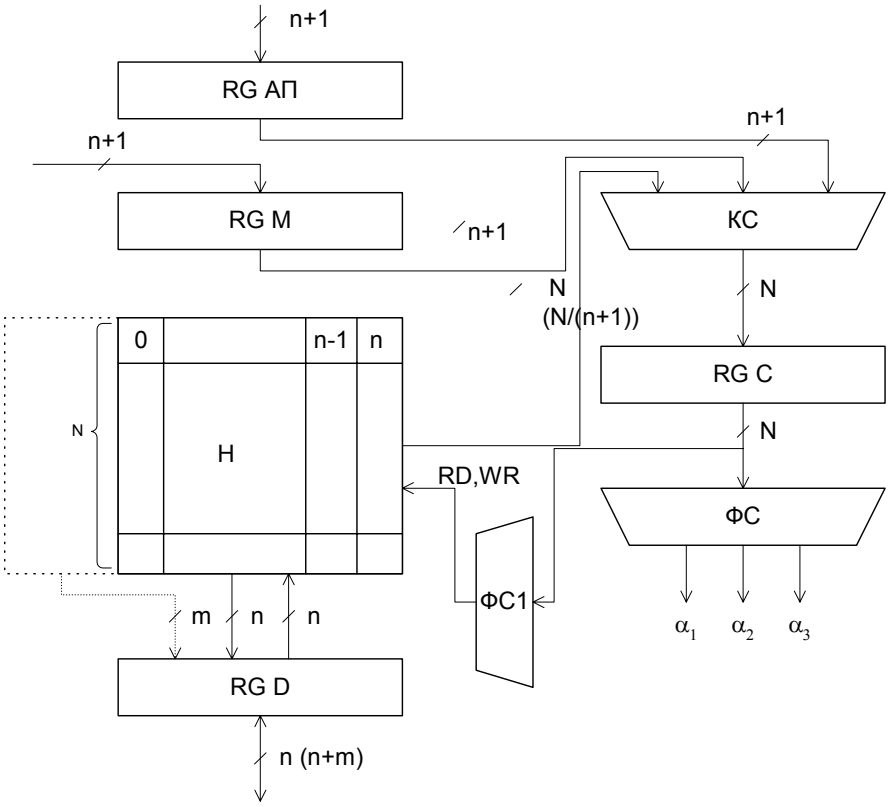


Рис. 2.6

Ассоциативная память применяется в основном в ВС, в которых решаются задачи распознавания образов, необходим быстрый поиск информации (например - в системах с аппаратной поддержкой БД). Также АП применяется в системах виртуальной памяти и кэш-памяти для определения необходимости подкачки страниц и для поиска страниц, подлежащих замене.

2.5. Системы памяти с расслоением

Принцип организации систем памяти с расслоением рассчитан на повышение быстродействия устройств памяти, состоящих из нескольких медленных устройств, за счет распределения адресного пространства между этими устройствами. (Напомним, что адресное пространство – это количество независимо адресуемых ячеек памяти). Адресное пространство делится таким образом, что соседние по адресам ячейки располагаются в разных физических устройствах. Логический адрес ячейки состоит из физического адреса внутри устройства (блока) и номера блока.

Расслоение памяти осуществляют двумя основными способами.

1. Повышение производительности памяти за счет одновременного считывания/записи соседних ячеек памяти из разных физических устройств по общей шине данных. Производительность увеличивается за счет параллельного подключения устройств и их одновременной работы на общую шину данных. Недостатком такого подхода является необходимость использования широкой

шины данных, часто – превышающей по ширине разрядность слов, используемых в системе.

2. Использование конвейера памяти. Конвейер строится из нескольких медленных устройств с большим временем доступа. Если система может обратиться с небольшой задержкой ($t_ц$) к нескольким медленным устройствам, то каждое из них начнет процесс выборки, и к тому моменту, как система закончит обращение к последнему устройству, первое уже выдаст считываемые данные на шину данных, затем – второе и т.д. В результате реальная производительность системы будет определяться не большим $t_{дост}$, а маленькой величиной $t_ц$.(см. Рис. 2.7)

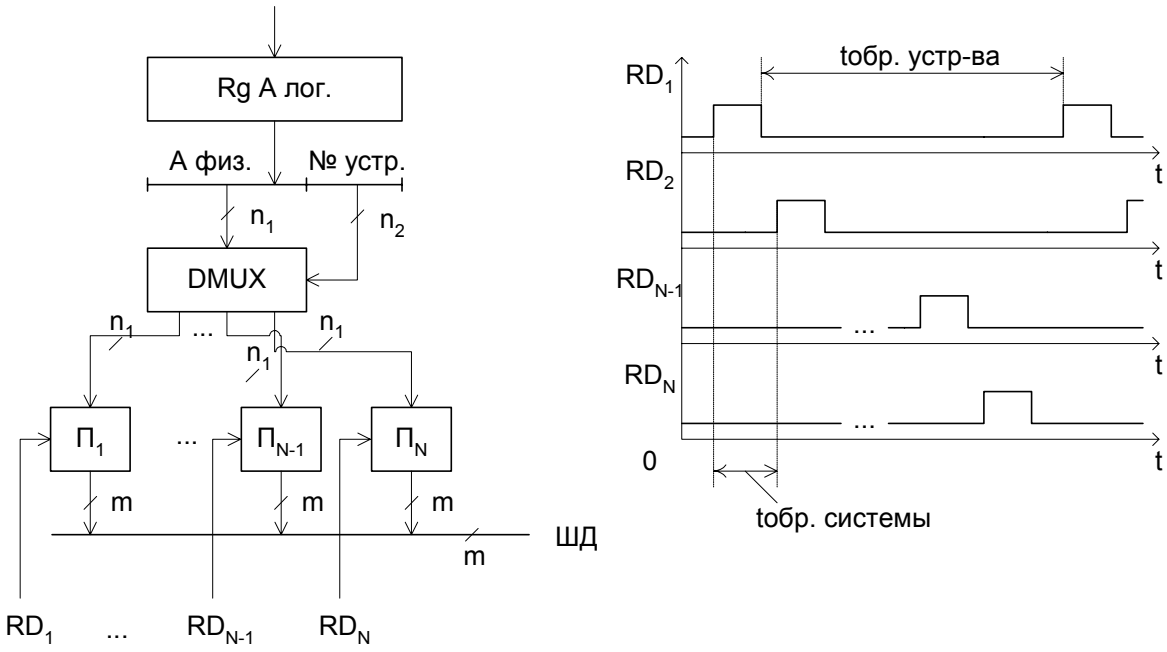


Рис. 2.7

Системы расслоения памяти применялись и применяются в различных ВС, в частности – в свое время в ПК типа IBM PC/XT, в супер-ЭВМ Cray –1,2, в других векторных процессорах. Похожие принципы конвейеризации работы памяти применяются в современных устройствах синхронной динамической памяти (SDRAM).

2.6. Понятие о виртуальной памяти

Виртуальная (от virtual – “кажущийся”) память (ВП) – это система организации памяти, при которой процессору (программе) предоставляется адресное пространство, превышающее физическое адресное пространство ОЗУ системы за счет внешней памяти. Задачей построения ВП является сведение к минимуму потерь производительности при вынужденном обращении к внешней памяти.

ВП может быть организована программно, программно-аппаратно и аппаратно. Как правило, в современных ВС программно-аппаратная организация

ВП заключается в использовании операционной системой аппаратной поддержки ВП, заложенной в процессорах общего назначения.

ВП может иметь страничную, сегментную или странично-сегментную организацию. При страничной организации память представляется совокупностью страниц фиксированной длины (2-16 Кбайт). При сегментной организации память представляет собой набор сегментов, то есть логически связанных блоков памяти различного размера.

Для виртуальной памяти большое значение имеет алгоритм подкачки, то есть способ замены страниц в ОЗУ на страницы во внешней памяти, к которым произошло обращение. При аппаратной организации ВП система подкачки использует ассоциативную память страниц. Стратегии замены страниц в ВП могут быть самыми различными:

1. Наиболее давнее использование (по времени)
2. Наиболее редкое использование.
3. По очереди (по принципу FIFO)
4. Случайным образом.
5. “Наилучший” выбор – гибкое сочетание различных стратегий.

2.7. Варианты организации КЭШ-памяти

Обособленным вариантом ВП можно считать т.н. кэш-память (от фр. «cache» – скрывать). Это вариант организации системы памяти, предназначенный для ускорения обмена между процессором и оперативной памятью. С виртуальной памятью кэш-память роднит общий принцип - ускорение за счет размещения наиболее активно используемых данных и кода в более быстрой памяти, но между ВП и кэш-памятью существует также множество различий, которые можно проиллюстрировать следующей таблицей:

Сравнение виртуальной и кэш-памяти. Таблица 2.1

Виртуальная память	Кэш-память
1. Организуется для ускорения обмена между процессором и внешней памятью (ОЗУ и ВнП)	1. Организуется для ускорения обмена между ЦП и ОЗУ
2. Обмен страницами по 2-16Кб	2. Обмен строками (сотни байт)
3. Ускорение до 1000 раз	3. Ускорение до 10 раз
4. При подкачке ЦП может переключаться на другую задачу	4. При подкачке ЦП ожидает ее завершения
5. Адресное пространство ВП равно сумме адресного пространства ОЗУ и ВнП	5. Адресное пространство кэш-памяти равно адресному пространству ОЗУ
6. В ОЗУ хранятся копии или оригиналы страниц ВП	6. В буферной памяти хранятся копии строк ОЗУ
7. ВП м.б. программно доступна	7. Кэш-память программно недоступна.

(Можно заметить, что под кэш-памятью иногда понимают не систему организации памяти, а саму буферную память (БП), используемую для ускорения обмена процессора с ОЗУ.)

Небольшое значение ускорения из-за использования кэш-памяти по сравнению со значительным ускорением при использовании виртуальной памяти можно объяснить большой разницей между временем доступа к дисковой памяти (10-ки микросекунд) и оперативной (10-ки наносекунд), и сравнительно небольшой – между временем доступа к оперативной памяти и к буферной памяти (наносекунды). Заметим, что буферная память в составе кэш-памяти обычно строится на базе быстродействующего статического ОЗУ на триггерах.

Системы кэш-памяти можно классифицировать следующим образом:

1. По способу отображения строк основной памяти на строки буферной памяти:
   - полностью ассоциативная кэш-память (любая строка основной памяти может размещаться в любой строке буферной памяти – самый дорогой и самый производительный вариант);
   - кэш-память с прямым отображением (каждая строка основной памяти может размещаться только в одной определенной строке основной памяти – самый простой и наименее производительный вариант);
   - частично-ассоциативная или множественно-ассоциативная кэш-память (компромиссный вариант, при котором основная память делится на множества строк, каждое множество отображается на группу строк в буферной памяти, при этом внутри группы действует принцип полной ассоциативности; при количестве групп = 1 получаем полностью ассоциативную кэш-память, при количестве групп = количеству строк – кэш-память с прямым отображением).
2. По способу переноса информации из кэш-памяти в основную (т.н. «своппинг»):
   - простой своппинг (Write Through - когда информация, записанная процессором в кэш-память, переносится в основную только при необходимости замены строки);
   - сквозной своппинг (Write Back - когда информация записанная процессором в кэш-память, одновременно переносится в основную, то есть кэш работает только на чтение; этот вариант менее производительный, но более надежный).

Рассмотрим подробнее варианты отображения строк основной оперативной памяти на буферную память на примере условной системы памяти с 16-ю строками в основной памяти (ОП) и 4 строками буферной памяти (БП). Такое небольшое количество строк выбрано для простоты изложения.

1. Полностью ассоциативная кэш-память (кэш-память с произвольным отображением). При таком варианте построения кэш-памяти в любой строке БП может располагаться любая строка из ОП (рис. 2.8). На рис. 2.8 : RGАлог – регистр логического адреса, RG D – регистр данных, хранящий всю строку из БП.

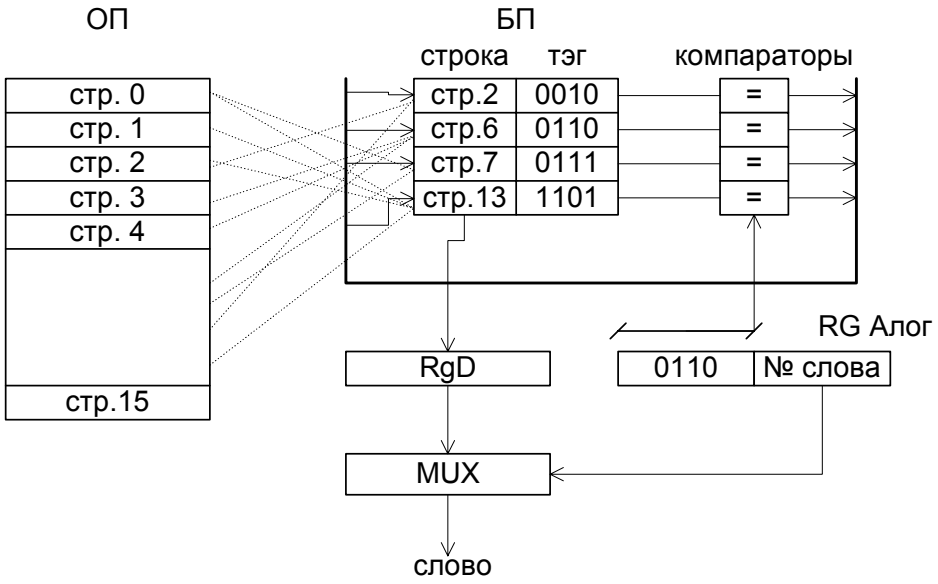


Рис. 2.8

Производительность системы с кэш-памятью, или величина ускорения при использовании кэш-памяти зависит в том числе от вероятности попадания искомой строки в БП :

Ркп = f (P_{hit}),

(P_{hit} – вероятность попадания, P_{miss} – вероятность промаха).

В свою очередь, вероятность попадания зависит как от объема буферной памяти (или – от соотношения объема БП и ОП), так и от количества комбинаций различных строк ОП, которые могут размещаться в БП. Для рассматриваемого варианта отображения такое количество комбинаций равно:

$K_1 = 2^N! / (2^N – 2^n)!$,

где 2^N – количество строк ОП, 2^n – количество строк БП (в данном случае N = 4, n =2, $2^N = 16$, $2^n = 4$).

2. Кэш-память с прямым отображением. При таком варианте построения кэш-памяти любая строка из ОП может располагаться только в одной конкретной строке БП (рис. 2.9).

Такой вариант является самым дешевым, но и самым медленным вариантом реализации, поскольку количество комбинаций различных строк ОП, которые могут размещаться в БП, существенно меньше, чем для полностью ассоциативной КП :

$K_2 = 2^N$.

3. Множественно-ассоциативная (частично-ассоциативная, ассоциативная по множеству) кэш-память. При таком варианте построения кэш-памяти (рис. 2.10) все множество строк основной памяти разбивается на несколько подмножеств (количество которых равно 2^s). Каждое подмножество отображается на группу строк в буферной памяти, внутри группы действует принцип полной ассоциативности, то есть любая строка из данного подмножества может располагаться в любой строке данной группы. Такой вариант является

промежуточным, компромиссным вариантом между полностью ассоциативной кэш-памятью и кэш-памятью с прямым отображением. Количество комбинаций:

$K_3 = 2^n (2^{N-S}!) / (2^{N-S} - 2^{n-S})!$,

При s=0 получаем полностью ассоциативную кэш-память (единственное подмножество), при s=n получаем вариант кэш-памяти с прямым отображением (количество подмножеств равно количеству строк в БП).

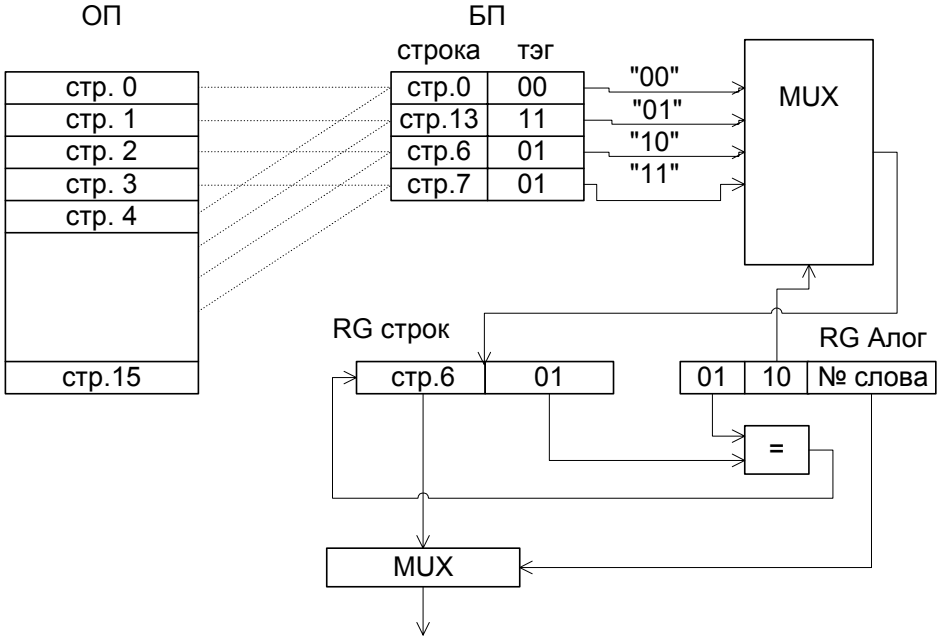


Рис. 2.9

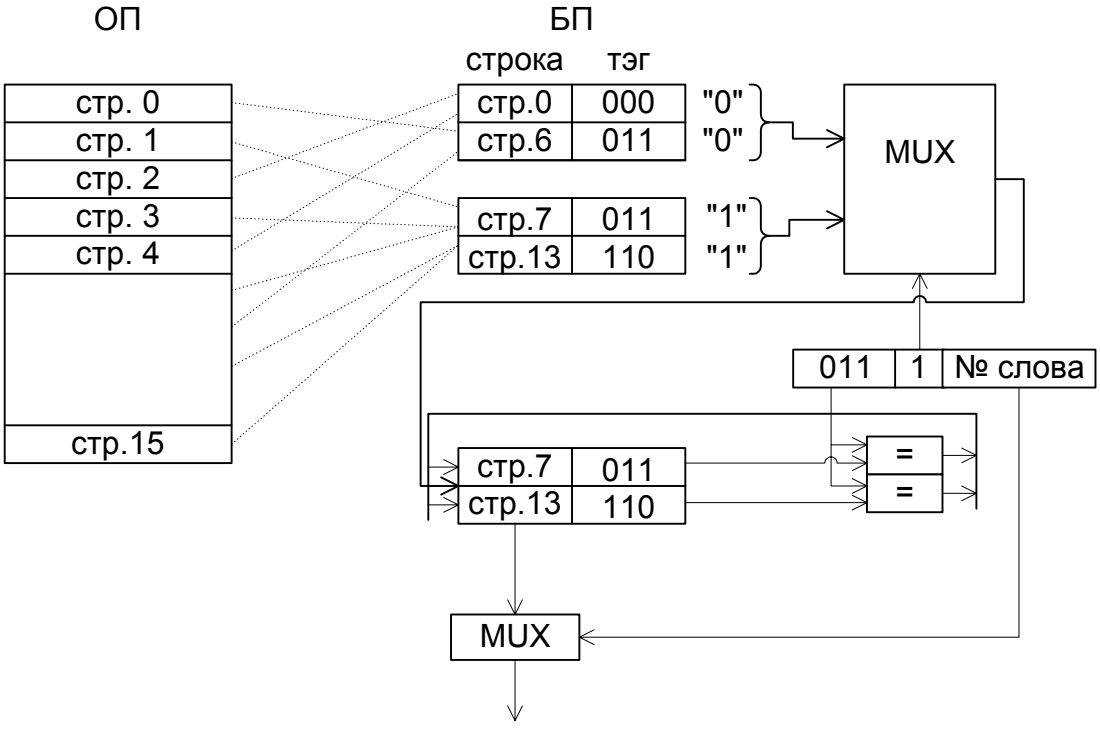


Рис. 2.10

В общем случае на ускорение кэш-памяти, которого она позволяет достичь, влияет ряд параметров:

- прежде всего, – размер кэш-памяти;
- способ отображения строк памяти;
- соотношение быстродействия устройств ОЗУ и буферной памяти;
- вариант свопинга.

Первые три параметра влияют на вероятность попадания слова в кэш-память (т.н. “cache hit”, при отсутствии попадания происходит кэш-промах – “cache miss”, приводящий к необходимости подкачки из основной памяти), которая непосредственно влияет на ускорение в системе с кэш-памятью.

Эффективное время обращения к кэш-памяти:

$$t_{обрКП} = t_{ПАП} + P\, t_{обрБП} + (1-P)\,(t_{ПАП} + t_{обрБП} + 2\, t_{обрОП})$$

где $t_{ПАП}$ − время поиска адресного признака ;

P – вероятность попадания в кэш ;

$t_{обрБП}$ − время обращения к буферной памяти;

$t_{обрОП}$ − время обращения к основной оперативной памяти.

В многопроцессорных системах с общей (разделяемой) памятью, в которых используется локальная для каждого процессорная кэш-память (буферная память), возникает проблема обеспечения непротиворечивого соответствия информации в разделяемой ОП и локальных копиях строк ОП в различных локальных блоках БП, известная как *проблема когерентности кэшей*.

В общем случае проблема сводится к тому, что запись одним процессором информации в свою буферную память не сразу приводит к изменению соответствующей ячейки в ОП, и, соответственно, другие процессоры, обращающиеся к этой ячейке ОП, либо – к ее копиям в своих модулях БП, видят «старую» информацию. В особенности это влияет на системы, использующие систему «почтовых ящиков» (ячеек ОП) для обмена заданиями между процессорами.

В таких системах могут использоваться различные методы разрешения указанной проблемы:

1 – запрещение переноса в кэш-память «почтовых ящиков» и другой служебной информации, используемой при обмене;

2 – фиксирование попадания в кэш-память подобных ячеек и их принудительное синхронное обновление во всех локальных копиях на аппаратном уровне;

3 – ограничение на максимальное количество чтений ячеек кэш-памяти (БП), подкачка из ОП при достижении максимума;

4 – информирование всех процессоров о попадании разделяемой информации в чью-либо БП.

5 – применение в многопроцессорных системах кэш-память со сквозным своппингом (сквозной записью).

3. ОРГАНИЗАЦИЯ ПРОЦЕССОРОВ

3.1. Назначение и классификация процессоров

Процессор – устройство, осуществляющее процесс автоматической обработки данных и программное управление этим процессом. Процессоры можно классифицировать, например, по следующим признакам:

1) По используемой системе счисления:
   - работающие в позиционной системе счисления;
   - работающие в непозиционной системе счисления (например, СОК).
2) По способу обработки разрядов:
   - с параллельной обработкой разрядов;
   - с последовательной обработкой;
   - со смешанной обработкой (последовательно-параллельной).
3) По составу операций:
   - процессоры общего назначения;
   - проблемно-ориентированные;
   - специализированные.
4) По месту процессора в системе:
   - центральный процессор (ЦП);
   - сопроцессор;
   - периферийный процессор;
   - канальный процессор (контроллер канала ввода/вывода);
   - процессорный элемент (ПЭ) многопроцессорной системы.
5) По организации операционного устройства (ОУ):
   - с операционным устройством процедурного типа (I-процессоры, M-процессоры) с преимущественно микропрограммным управлением;
   - процессоры с блочным операционным устройством;
   - процессоры с конвейерным операционным устройством (с арифметическим конвейером) (последние два варианта предусматривают аппаратную реализацию большинства операций ОУ).
6) По организации обработки адресов:
   - с общим операционным устройством;
   - со специальным (адресным) операционным устройством.
7) По типу операндов:
   - скалярный процессор;
   - векторный процессор;
   - с возможностью обработки и скалярных, и векторных данных.
8) По логике управления процессором:
   - с жесткой логикой управления;
   - с микропрограммным управлением.

9) По составу (полноте) системы команд:
   - RISC (Reduced Instruction Set computer – компьютер с сокращенным набором команд);
   - CISC (Complete Instruction Set Computer– компьютер с полным набором команд);
   - CISC – процессор с внутренними RISC-подобными инструкциями.
10) По организации управления потоком команд / способу загрузки исполнительных устройств:
   - с последовательной обработкой команд;
   - с конвейером команд;
   - суперскалярные процессоры;
   - процессоры с длинным командным словом (VLIW – Very Long Instruction Word) и т. д.

Как всякая классификация, приведенная выше классификация не может считаться полной, так как количество типов процессоров достаточно велико и по своим архитектурам процессоры весьма многообразны.

3.2. Логическая организация процессора общего назначения

Схема, отражающая логическую организацию некоего усредненного процессора общего назначения, представлена на рис. 3.1. В основе структуры процессора лежит взаимосвязь операционной и управляющих частей, что соответствует модели цифрового автомата, предложенной академиком Глушковым /13-14/. Операционные устройства процессора (средства обработки, исполнительные устройства) включают в общем случае ОУ с фиксированной запятой (целочисленное ядро, АЛУ), ОУ с плавающей запятой (числовой сопроцессор или ядро с плавающей запятой), устройство для реализации десятичной арифметики и возможно – устройства для обработки строк и массивов.

Отметим, что в некоторых процессорах отдельно реализуется специальное устройство для вычисления адресов (так называемая разнесенная - decoupled - архитектура), в других процессорах вычисление адресов происходит в общем операционном устройстве. Операционное устройство неразрывно связано с наиболее быстродействующей памятью ВМ – с локальной регистровой памятью процессора. Выделение регистров в отдельный блок на схеме призвано подчеркнуть самостоятельное значение, которое приобретают регистры в универсальных процессорах – они не просто являются частью операционных устройств, а используются для хранения различной информации как при обработке, так и при вводе-выводе. Целочисленные регистры объединяются в блок регистров общего назначения (РОН), регистры с плавающей запятой – в отдельный блок (в некоторых процессорах эти многоразрядные регистры используются и как векторные регистры в специальных режимах, в других векторные регистры вынесены в отдельный блок).

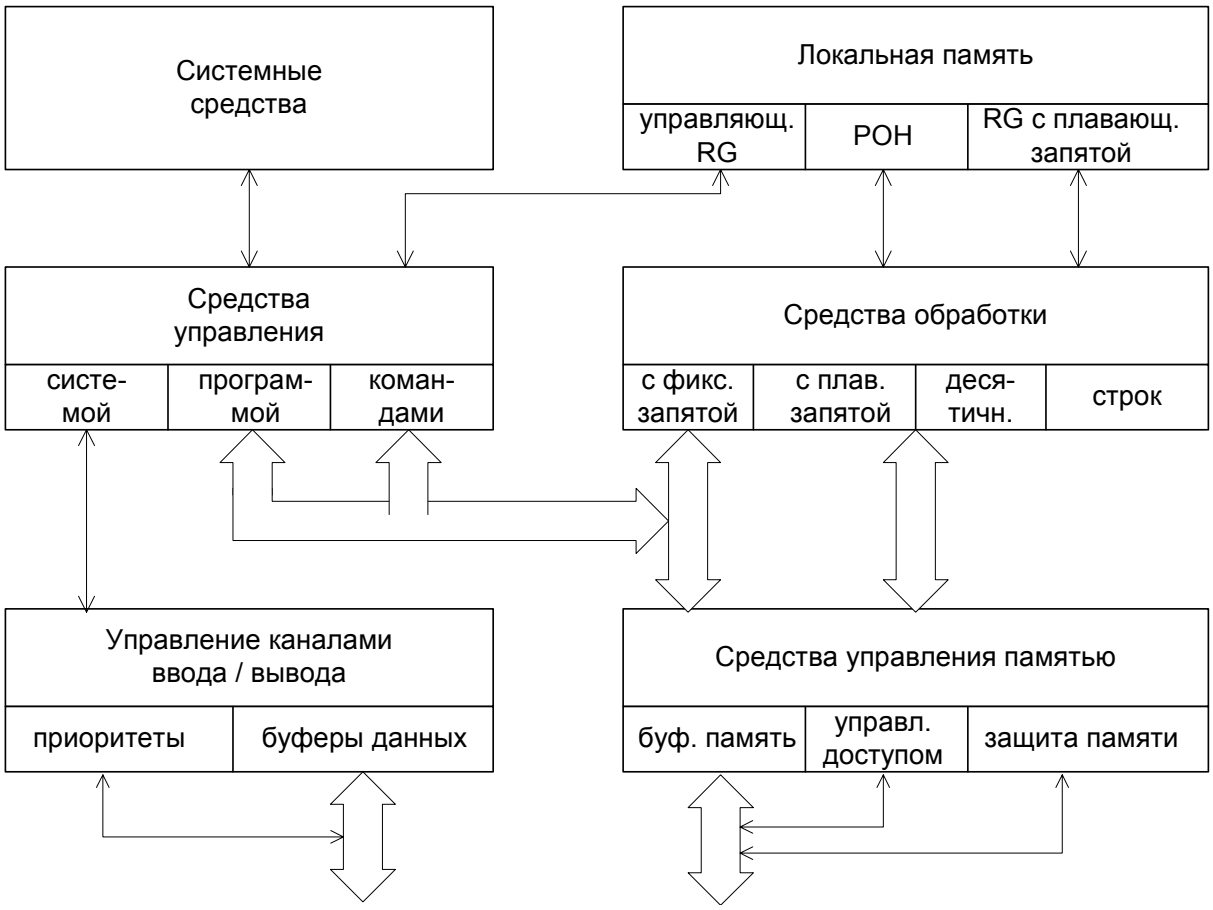


Рис. 3.1

Кроме упомянутых регистров можно выделить набор специальных управляющих регистров, используемых для управления режимами работы процессора, функционированием его различных подсистем, управления памятью и т.д. Средства управления процессором (или – устройство управления) выполняют разнообразные функции, которые включают: управление системой, программой и командами. Управление системой подразумевает управление прерываниями, остановом и запуском процессора, обеспечение отладочного режима и вообще выбор режимов работы процессора и т.д. Управление программой включает обеспечение выполнения ветвлений и циклов, вызовов и возвратов из подпрограмм и т.д. Средства управления командами обеспечивают выполнение машинных циклов работы процессора, то есть – выборки команды, ее дешифрации, собственно управления выполнением команды, управление записью результатов. В данной подсистеме могут реализовываться собственно управляющие автоматы, отвечающие за реализацию алгоритмов, заложенных в командах процессора (то есть за реализацию микрокода процессора). В некоторых случаях подобные подсистемы относят к УУ, в других – включают в состав собственно средств обработки, то есть рассматривают как часть АЛУ и числового сопроцессора, что в принципе не так важно.

Помимо выполнения операций, вычисления адресов и программного управления этими процессами, процессор должен содержать средства для обеспечения интерфейса как с оперативной памятью, так и с внешними

устройствами (интерфейсы ввода-вывода). В состав интерфейса с памятью могут включаться буферная память (кэш-память), средства управления доступом и защиты памяти. Интерфейс с каналами ввода-вывода включает буферы данных, систему управления приоритетами, входящую в подсистему прерываний процессора, и т.д.

Под системными средствами понимают встроенные схемы синхронизации, возможно – таймеры, какие-то дополнительные схемы управления, сброса и т.д.

3.3. Операционные устройства процессоров

3.3.1. Операционные устройства процедурного типа и с жесткой структурой. Понятие об I-процессорах и M-процессорах

Операционные устройства процессоров могут строиться с большей или меньшей степенью универсальности, могут быть более простыми, универсальными, требующими большого объема микрокода для реализации всех необходимых алгоритмов операций, либо – более сложными и специализированными, но за счет этого – более производительными и не требующими большого объема управляющего микрокода. Первые устройства можно назвать устройствами процедурного типа, так как они требуют для реализации какого-либо алгоритма арифметической операции выполнения последовательности действий, заданной во времени (то есть процедуры).

Устройства второго типа, рассчитанные на аппаратную реализацию алгоритмов вычислений, можно назвать устройствами с жесткой структурой. (Отметим, что гибкость устройств первого типа заключается не в возможности перестройки их структуры, а в возможности выполнения на заданной структуре большего числа различных алгоритмов.) Примером устройств процедурного типа могут являться, до некоторой степени, устройства для выполнения косвенного умножения. Такие устройства после небольшой доработки могут быть использованы и для реализации других операций (алгоритмов), например, для обычного сложения со знаком, для выполнения деления или операций с плавающей запятой. В предельном случае наиболее универсальной схемой может являться обычный накапливающий сумматор, дополненный схемами выполнения логических операций. С другой стороны, специализированный аппаратный умножитель, например, матричный (матричные умножители подробнее рассматриваются в следующем пункте), является примером устройства с жесткой структурой, рассчитанного только на выполнение конкретной операции, зачастую – определенной разрядности и в определенной кодировке. Для создания более или менее универсального ОУ необходимо иметь набор таких схем для всех требуемых операций, либо – сочетание нескольких специализированных устройств с одним универсальным.

Операционные устройства процедурного типа могут быть построены различными способами. Примером процессоров с более жестким принципом построения операционной части процедурного типа являются так называемые

I-процессоры, у которых за определенными регистрами закреплены определенные операции (рис.3.2). На рисунке 3.2 ША и ШД – соответственно шины адреса и данных, Acc – аккумулятор, КС – комбинационные схемы, ТП – триггеры признаков, УУ – устройство управления. Разные регистры соединены с разными операционными элементами (КС) и по-разному соединены друг с другом. Такое разнесение операций по регистрам за счет наличия нескольких операционных элементов в схеме позволяет распараллелить выполнение некоторых вычислений и тем самым повысить производительность. С другой стороны, такая организация подчас лишена необходимой гибкости и требует частых пересылок информации между регистрами.

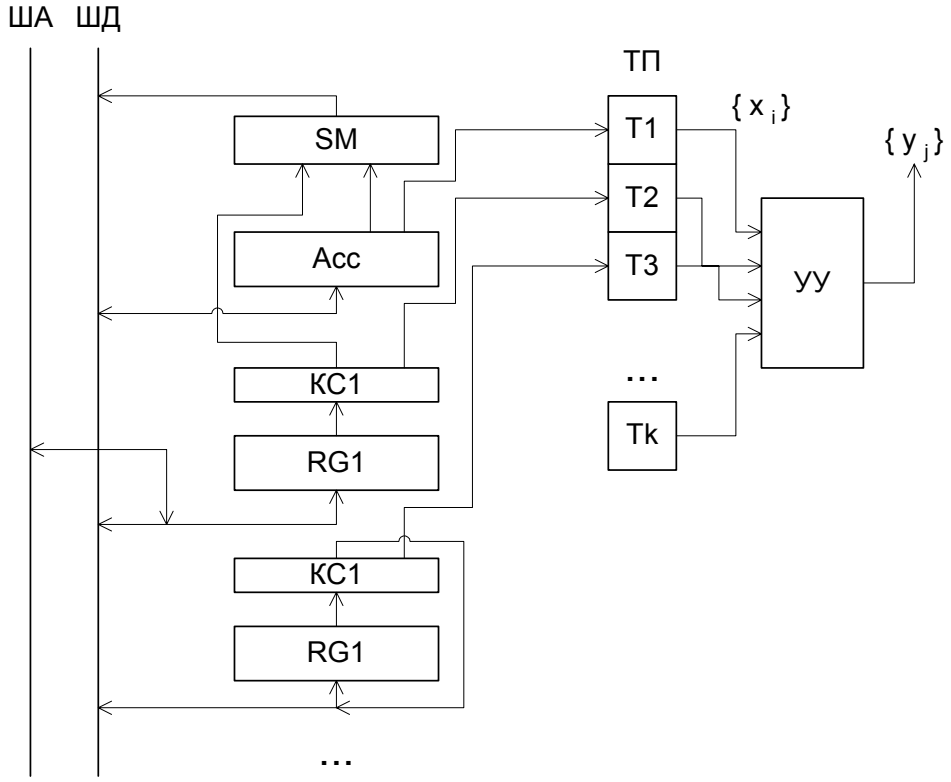


Рис. 3.2

В процессорах с магистральной архитектурой (процессоры М-типа, или процессоры с общим АЛУ) имеется одно обрабатывающее устройство - сумматор, либо АЛУ (например, табличное), с которым связаны все регистры из блока РОН (рис. 3.3).

Регистры являются в данном случае равноправными, каждая пара регистров может участвовать в любой операции. АЛУ связано с регистрами тремя магистралями - магистрали А и В служат для подачи операндов в АЛУ, а магистраль С – для записи результата в выбранный регистр из блока РОН.

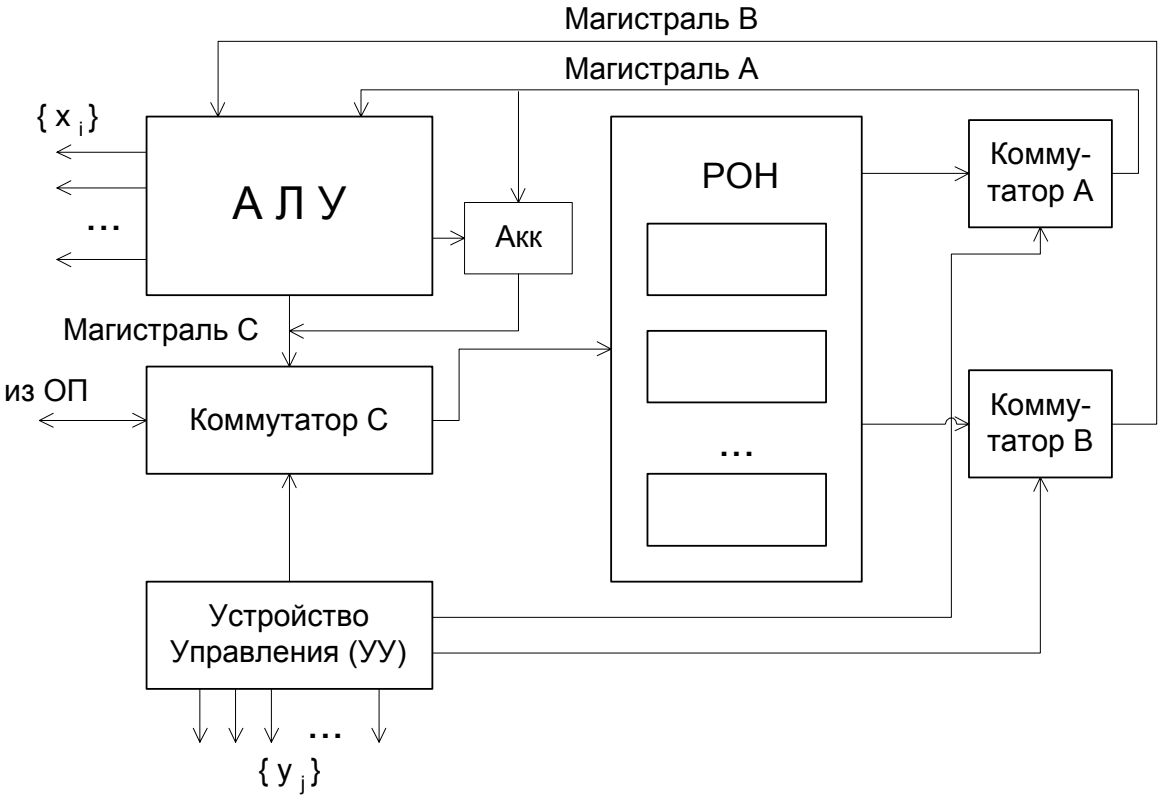


Рис. 3.3

Иногда один из регистров все же выделяется как особый, в котором могут выполняться специальные операции, недоступные для других регистров. В ряде случаев этот регистр всегда является приемником результата (а иногда – обязательно и одним из операндов). Тогда такой регистр называют аккумулятором, а процессор называют процессором на базе аккумулятора. В принципе, в АЛУ такого процессора можно разместить какое-то количество специализированных арифметических устройств жесткой структуры, тогда полученное ОУ будет чем-то промежуточным между процедурным и жестким.

3.3.2. Блочные операционные устройства

Для повышения производительности процессора при выполнении операций его операционное устройство может строиться по блочному принципу. В таких *блочных ОУ* реализуется несколько функционально независимых исполнительных устройств, выполняющих различные операции (или различные группы операций, например, три блока целочисленного сложения, два – целочисленного умножения, по одному блоку деления, сложения и умножения с плавающей запятой и т.д.).

Эти устройства работают параллельно, обрабатывая каждое свои операнды. Управление этими устройствами осуществляется с помощью так называемых длинных командных слов (Very Long Instruction Word - VLIW) . Командные слова включают инструкции для каждого их исполнительных устройств, а также операнды или указатели на них. (О процессорах VLIW также см. пункт 3.5.5)

Преимуществом блочных ОУ является более высокая производительность, достигаемая за счет распараллеливания вычислений. В то же время, использование таких устройств не всегда эффективно, поскольку не всегда есть возможность загрузить все исполнительные устройства в каждом такте, в результате часть из них простаивает. Более эффективными часто оказываются конвейерные операционные устройства, поскольку конвейеризовать вычисления в ряде случаев проще, чем распараллелить, что связано с повторением однотипных вычислений в алгоритмах.

3.3.3. Конвейерные операционные устройства

Для конвейеризации вычислений необходимо:

- разбить вычисления на последовательность одинаковых по времени этапов;
- реализовать каждый этап аппаратно в виде ступени конвейера;
- обеспечить фиксацию промежуточных результатов вычислений на выходе каждой ступени в регистрах-защелках.

Напомним, что эффективность конвейера будет тем выше, чем больше задач будет поступать на его вход.

Типичным примером конвейерных операционных устройств могут служить так называемые матричные умножители. Свое название они получили, во-первых, потому, что включают фактически матрицу операционных элементов (сумматоров), а во-вторых, поскольку одной из наиболее очевидных сфер их применения является умножение матриц.

Рассмотрим процесс умножения двух двоичных четырехразрядных положительных чисел:

$$
\begin{array}{cccccccc}
 & & & & a_3 & a_2 & a_1 & a_0 \\
 & & & \times & b_3 & b_2 & b_1 & b_0 \\
\hline
 & & + & & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 \\
 & + & & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & \\
 + & & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & \\
+ & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & & \\
\hline
c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0
\end{array}
$$

По косвенной схеме умножения на устройстве с одним сумматором и набором регистров для реализации этого умножения необходимо в общем случае выполнить 4 шага, на каждом их которых выполняется умножение А на очередной разряд b_i , сложение А b_i с текущей суммой частичных произведений и сдвиг новой полученной суммы на 1 разряд вправо. Таким образом, время на выполнение этого умножения можно приближенно оценить как :

$$T_{умн} = 4(t_{\&} + 4*t_{sm} + t_{sh}) ,$$

где $t_{\&}$ -задержка на 1 логическом вентиле (при умножении А на b_i),

$4t_{sm}$-задержка на сложение при использовании сумматора с последовательным переносом, t_{sm} – задержка одноразрядного полного двоичного сумматора, которую можно принять равной $2t_{\&}$,

t_{sh} -задержка на 1 сдвиг, которую также можно приравнять $2t_{\&}$.

Тогда время умножения = 44 $t_{\&}$. С другой стороны, поскольку все произведения A b_i в принципе можно рассчитать параллельно, сдвиги также можно задать жестко, а переносы при сложении можно учесть только при завершении сложения всех 4 слагаемых (частичных произведений), вместо того, чтобы рассчитывать их на каждом шаге, процесс умножения можно значительно ускорить, если реализовать схему матричного умножителя, представленную на рис. 3.1. Это схема умножителя Брауна. На схеме не показаны вентили, необходимые для получения всхе частичных произведений, а также фиксаторы, необходимые на выходе каждой линейки сумматоров. Каждая линейка сумматоров представляет собой так называемый сумматор с сохранением переноса (ССП), который широко применяется в различных арифметических устройствах.

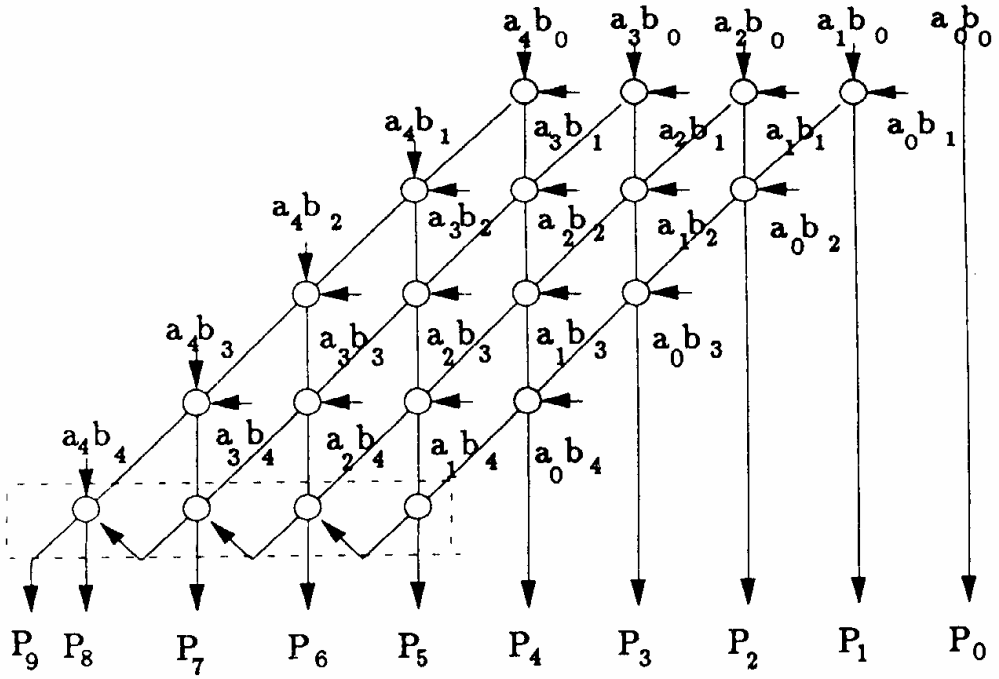


Рис. 3.4

Пунктиром на рисунке обведен параллельный сумматор, который может быть реализован, как показано на рисунке, то есть как сумматор с последовательным переносом, или – по схеме с ускорением переноса. Время умножения на подобном уножителе :

$T_{умн} = t_{\&} + (n + m - 2)*t_{sm}$,

где n – разрядность множимого, m – разрядность множителя.

В формуле не присутствуют затраты на сдвиги, так как они залдаются жестко путем соединений линеек сумматоров, кроме того, считаем, что все частичные произведения формируются за 1 логическое умножение. Для нашего случая время на умножение оказывается равным 13 $t_{\&}$. Таким образом, быстродействие умножителя по сравнению с обычной схемой примерно в 3 раза выше. Кроме того, умножитель может работать в режиме конвейера. В данном случае число его ступеней равно 6 (так как в сумматоре с последовательным переносом придется организовывать три отдельные ступени). Пиковая производительность конвейера при полной загрузке – 1 результат за $2t_{\&,}$ то есть в 20 раз выше, чем в обычной схеме. Такой выигрыш достигается за счет дополнительных аппаратных затрат, которые выше, чем в первом случае примерно в 4-5 раз.

В умножителе Брауна используются несколько основных способов повышения производительности:

- рапспараллеливание вычислений (одновременное вычисление всех Ab_i);
- конвейеризация вычислений (цикл умножения разворачивается в последовательность ступеней, межразрядные переносы сохраняются и передаются на следующую ступень);
- аппаратная реализация и специализация вычислений позволяет избежать расходов на сдвиг, который задается жестко, сохранение переноса также диктуется выбранным для аппаратной реализации алгоритмом.

Как уже упоминалось, основным элементом матричного умножителя является сумматор с сохранением переноса (ССП или Carry Save Adder - CSA). Его используют не только в умножителях, но и везде, где необходимо ускорить сложение N чисел. Так, на рис. 3.5. показан сумматор для сложения 3 чисел на базе ССП. Остановимся на принципе построения подобных устройств. Полный сумматор (ПС) позволяет складывать 3 одноразрядных числа. Обычно в качестве третьего слагаемого выступает перенос, поступающий либо с предыдущего сумматора, либо со схемы передачи переноса. Но если в качестве третьего слагаемого использовать соответствующий разряд третьего n-разрядного числа и не передавать перенос в следующий одноразрядный сумматор, то на выходе сумматора сформируется сумма в данном разряде и перенос.

На выходе линейки таких сумматоров формируются два числа - собственно сумма разрядов трех n-разрядных слагаемых и сумма переносов при сложении этих слагаемых. Сумма этих двух чисел и представляет собой значение суммы трех слагаемых:

$$S = X + Y + Z = S_{xyz} + C_{xyz}.$$

Линейка полных сумматоров, обведенная на рис. 3.5 пунктиром - это и есть сумматор с сохранением переноса (ССП). Данная схема имеет 3 входа и два выхода (имеются в виду n-разрядные входы и выходы), поэтому в литературе можно встретить для нее обозначение $ССП_{3\text{-}2}$.

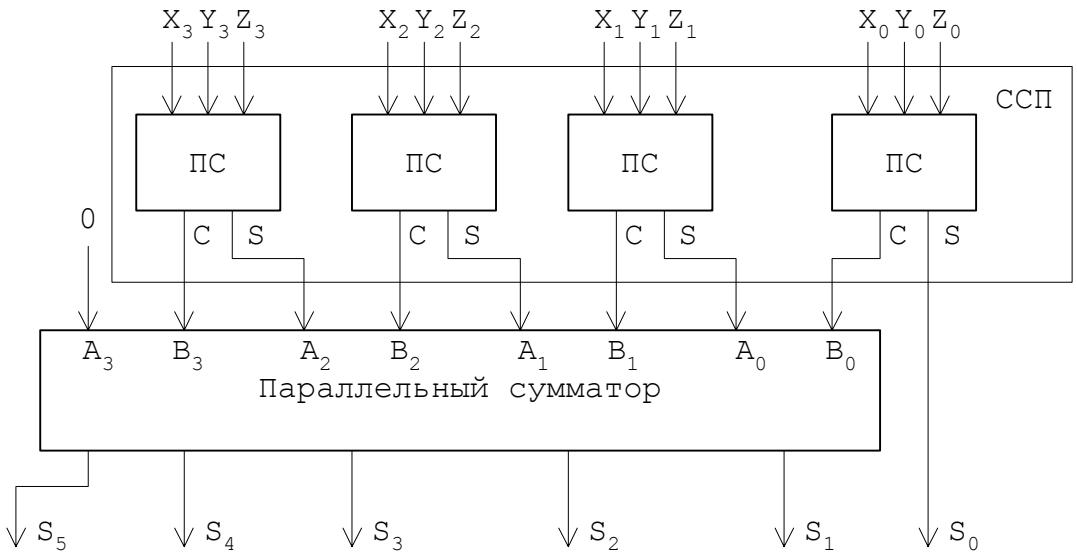


Рис. 3.5

Если подать два полученных числа на обычный параллельный сумматор, то на выходе мы получим сумму 3 чисел. Если использовать не один $ССП_{3-2}$, а дерево таких сумматоров, как показано на рис.3.6 ($ССП_{8-2}$), то выполняется сложение 8 чисел, и так далее - для N чисел мы используем схему $ССП_{N-2}$. Фактически мы имеем схему, похожую на пирамидальную, но с одной общей схемой передачи и ускорения переноса.

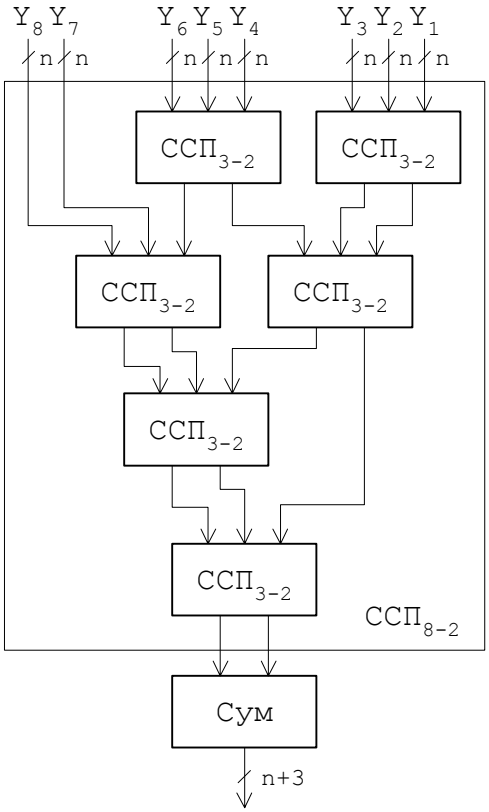


Рис. 3.6.

Ускорение схемы на базе ССП по сравнению с пирамидальным включением сумматоров зависит от времени задержки параллельного сумматора со схемой ускоренного переноса (СУП):

$$K_{уск} = \frac{t_{сум} \log_2 N}{t_{сум} + t_{зс} \log_{3/2} N},$$

где $t_{сум}$ - время задержки параллельного сумматора с СУП,

$t_{зс}$ - задержка полного одноразрядного сумматора.

При этом необходимо отметить, что для большинства вариантов СУП ускорение схемы с ССП по сравнению с пирамидальной возрастает при увеличении разрядности слагаемых, так как соответственно растет $t_{сум}$, а $t_{зс}$ не меняется.

На базе быстродействующего сумматора на N чисел, аналогичного представленному на рис. 3.6, можно построить древовидный умножитель Уоллеса. В таком

устройстве умножение выполняется в 2 этапа – на первом формируются все частичные произведения вида $A \cdot b_i \cdot 2^i$, на втором – полученные N частичных произведений (где N – количество разрядов множителя без учета знаковых) складываются на сумматоре с $ССП_{N-2}$, как показано на рис. 3.7 на примере умножения на 8-и разрядный множитель. По сравнению с умножителем Брауна мы имеем выигрыш в быстродействии за счет использования большего количества ССП, что позволяет в большей степени распараллелить процесс сложения частичных произведений.

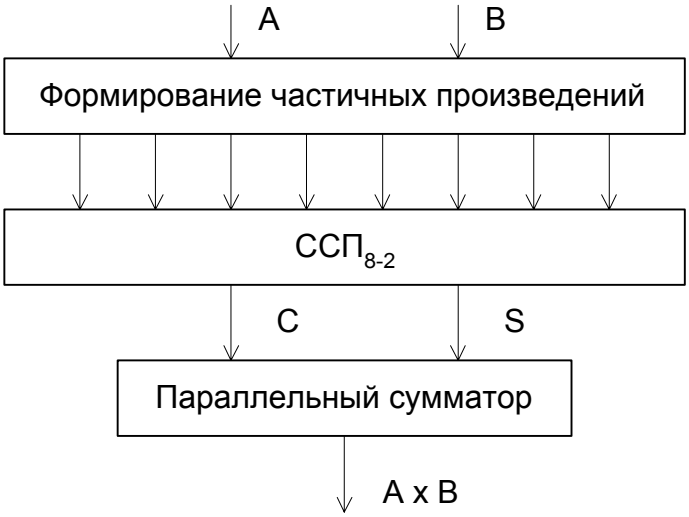


Рис. 3.7

Конвейерные ОУ могут использоваться самостоятельно, но чаще являются составной частью ОУ процедурного типа, либо - блочных ОУ как аппаратные ускорители выполнения операций.

3.4 Архитектура системы команд. RISC и CISC процессоры

Под архитектурой системы команд (ISA – Instruction Set Architecture) понимают состав и возможности системы команд, общий взгляд на систему команд (СК) и связанную с ней микроархитектуру процессора с точки зрения программиста. Во многом именно архитектура СК определяет трактовку архитектуры компьютера вообще как «…абстрактного представления о вычислительной машине с точки зрения программиста».

Исторически первые микропроцессоры, появившиеся в 70-х годах XX века, имели относительно простую систему команд, что объяснялось небольшими возможностями интегральной схемотехники. По мере увеличения степени интеграции ИМС разработчики МП старались расширять систему команд и делать команды более функциональными, «семантически нагруженными». Это объяснялось, в частности, двумя моментами – во-первых, требованиями экономить память для размещения программ, оставлять больше памяти под данные и т.д., а во-вторых – возможностью реализовать внутри кристалла процессора сложные инструкции быстрее, чем при их программной реализации.

В результате появились процессоры с большими наборами команд, причем команды эти также зачастую являлись достаточно сложными. В последствии эти МП назвали CISC – от Complete Instruction Set Computer – компьютер с полным набором команд или Complex ISC – со сложным набором команд. Типичным примером CISC-процессоров являются процессоры семейства x86 корпорации Intel и ее конкурентов (а также Motorola 68K и другие).

Наряду с отмеченными преимуществами процессоры CISC обладали и рядом недостатков, в частности – команды оказывались сильно неравнозначными по времени выполнения (разное количество тактов), плохо конвейеризовывались, требовали сложного (и длительного) декодирования и выполнения.

Для повышения производительности стали использовать жесткую логику управления, что отразилось на регулярности и сложности кристаллов (нерегулярные кристаллы менее технологичны при изготовлении). На кристалле оставалось мало места для РОН и КЭШ.

Кроме того, исследования показали, что производители компиляторов и просто программисты не используют многие сложные инструкции, предпочитая использовать последовательность коротких.

Разработчики подошли к концепции более простого и технологичного процессора с некоторым откатом назад – к простым и коротким инструкциям. С конца 70-х до середины 80-х годов появляются проекты таких процессоров Стэндфордского университета и университета Беркли (Калифорния) – MIPS и RISC.

В основу архитектуры RISC (от Reduced Instruction Set Computer – компьютер с сокращенным набором команд) положены, в частности, принципы отказа от сложных и многофункциональных команд, уменьшения их количества, а также концентрация на обработку всей информации преимущественно на кристалле процессора с минимальными обращениями к памяти.

Основные особенности архитектуры RISC:

1. Уменьшение числа команд (до 30-40).
2. Упрощение и унификация форматов команд.
3. В системе команд преобладают короткие инструкции (например, часто в СК отсутствуют умножения).
4. Отказ от команд типа память-память (например, MOVSB в x86).
5. Работа с памятью сводится к загрузке и сохранению регистров (поэтому другое название RISC - Load-Store Architecture - архитектура типа «загрузка-сохранение»).
6. Преимущественно реализуются 3-х адресные команды, например :
   add r1, r2, r3 – сложить r2 с r3 и поместить результат в r1.
7. Большой регистровый файл - до 32-64 РОН.
8. Предпочтение отдается жесткой логике управления.

Преимущества архитектуры RISC:

1. Облегчается конвейерная, суперскалярная и другие виды параллельной обработки, планирование загрузки, предвыборка, переупорядочивание и т.д.
2. Более эффективно используется площадь кристалла (больше памяти – РОН, кэш).
3. Быстрее выполняется декодирование и исполнение команд – соответственно, выше тактовая частота.

Примерами семейств процессоров с RISC-архитектурой могут служить DEC Alpha , SGI MIPS, Sun SPARC и другие.

Большинство современных суперскалярных и VLIW-процессоров (в т.ч. и Intel) либо имеют архитектуру RISC, либо реализуют похожие на RISC принципы, либо – поддерживают CISC-инструкции, но внутри транслируют их в RISC-подобные команды для облегчения загрузки конвейеров и решения других задач.

3.5. Устройства управления процессоров

3.5.1 Назначение и классификация устройств управления

Как уже упоминалось ранее, устройство управления процессора отвечает за выполнение собственно команд процессора, включая основные этапы (загрузка, декодирование, обращение к памяти, исполнение, сохранение результатов), управление выполнением программ (организация ветвлений, циклов, вызов подпрограмм, обработка прерываний и др.), а также – управляет работой процессора в целом.

Устройства управления классифицируются в зависимости от типа процессора, или – типа управления исполнением команд, который в нем применяется :

- устройства управления процессора общего назначения или – спецпроцессора;
- устройства управления с поддержкой конвейера команд, без такой поддержки, или – с поддержкой многопотокового конвейера (в суперскалярных процессорах), а также – устройство управления процессора с длинным командным словом;
- устройство управления с упорядоченным исполнением команд, неупорядоченным исполнением, выдачей, или завершением команд (с поддержкой динамической оптимизации).

Кроме того, можно выделить устройства управления, построенные на базе памяти микропрограмм (с программируемой логикой), либо – на базе триггерных автоматов (с жесткой логикой).

Мы рассмотрим организацию устройства управления (а вернее – пары устройство управления – операционное устройство) для очень простого учебного RISC – процессора, а затем – рассмотрим способы ускорения работы процессора, основанные на конвейеризации и распараллеливании команд.

3.5.2 Архитектура простого RISC - процессора

Рассмотрим архитектуру простого RISC-процессора на примере некоторого процессора ARC («A RISC Computer») с системой команд, являющейся подмножеством системы команд процессора SPARC. / 16 /

Процессор является 32-разрядным (то есть обрабатывает 32-битовые слова в своем АЛУ), разрядность его команд – также 32 бита. Адресуемая память - 2^{32} байт или 2^{30} команд.

Большинство команд процессора – трехадресные, следующего формата:

opp rd, rs1, rs2 ; где opp – код команды, rs1,2 – регистры источники, ; rd – регистр приемник, или

opp rd, rs1, imm13 ; где imm13 – непосредственное значение 13 бит.

Все команды можно разделить на следующие группы:

1. Команды работы с памятью : ld (load - загрузка) и st (store – сохранение).
2. Логические команды : and, or, nor, srl (сдвиг), sethi rd, imm22 (установка старших 22 бит регистра в заданные значения).
3. Арифметическая команда : add (сложение).
4. Команды управления: ветвления be, bneg, bcs, bvs, ba (безусловный переход), все ветвления в формате be imm22 (относительное смещение), команда call imm30 – вызов подпрограммы, jmpl (ret) – возврат из подпрограммы.

Таблица 3.1

F_3	F_2	F_1	F_0	Operation	Changes Condition Codes
0	0	0	0	ANDCC (A, B)	yes
0	0	0	1	ORCC (A, B)	yes
0	0	1	0	NORCC (A, B)	yes
0	0	1	1	ADDCC (A, B)	yes
0	1	0	0	SRL (A, B)	no
0	1	0	1	AND (A, B)	no
0	1	1	0	OR (A, B)	no
0	1	1	1	NOR (A, B)	no
1	0	0	0	ADD (A, B)	no
1	0	0	1	LSHIFT2 (A)	no
1	0	1	0	LSHIFT10 (A)	no
1	0	1	1	SIMM13 (A)	no
1	1	0	0	SEXT13 (A)	no
1	1	0	1	INC (A)	no
1	1	1	0	INCPC (A)	no
1	1	1	1	RSHIFT5 (A)	no

Регистры процессора: 32 РОН, IR (instruction register - регистр команды), PC (program counter – программный счетчик), PSR (Program Status Register – слово состояния программы - 4 флага). Все регистры – 32- разрядные.

В процессоре поддерживаются следующие режимы адресации: непосредственная, регистровая, косвенная регистровая, косвенная регистровая по базе (индексная).

Адресная арифметика в процессоре реализуется на том же АЛУ, что и основные операции. АЛУ построено на таблицах истинности, а также включает программируемый нетактируемый сдвигатель на базе мультиплексора. АЛУ выполняет до 16 простых коротких арифметических или логических операций, приведенных в таблице 3.1. Форматы команд приведены в таблице 3.2.

Микроархитектура процессора представлена на рис. 3.8. На рисунке использованы следующие обозначения: **Data Section** – операционное устройство (ОУ); **Control Section** – устройство управления (УУ); **Main Memory** – основная память (ОП); **Scratchpad**-сверхоперативное ОЗУ (32 РОН %**r0**..%**r31**, 4 временных регистра %**temp0**..%**temp3,** регистры управления **ir, pc**); **C BUS MUX** – шинный мультиплексор С для выбора источника данных для регистра-приемника из памяти или с выхода АЛУ; в регистре команд **ir**: **rd** – адрес регистра-приемника, **rs1**, **rs2** – адреса регистров источников, **i** - флаг непосредственной адресации, **ops** – код операции; **MIR** – регистр микрокоманды (**РМК**); мультиплексоры **A, B, C** – выбирают адрес соответствующего регистра либо из **ir,** либо – из соответствующего поля **РМК** в зависимости от флагов

MUXA, **MUXB**, **MUXC; Control Store (CS)** – память микропрограмм (**ПМП**); **CSAI** – счетчик адреса микропрограммы; **CS Address MUX** – мультиплексор адреса микропрограммы (3 канала – **Next** – следующий адрес из CSAI, **Jump** - переход по адресу, указанному в РМК, **Decode** – переход к микро-подпрограмме реализации команды); **CBL** – логика управления ветвлением; %**psr** – регистр состояния программы, хранит 4 флага результата последней операции: **n**-netgative (отрицательное число), **z**-zero (ноль), **v**-overflow (переполнение), **c**-carry (перенос); **ACK** – подтверждение о готовности памяти для инкремента адреса микрокоманды; в РМК также отметим поля: **RD/WR** – чтение/запись памяти, **ALU** – код операции АЛУ, **JUMP ADDR** – адрес перехода в микропрограмме.

Операционная часть ARC соответствует операционной части М-процесора. Управляющая часть напоминает структуру управляющего автомата с программируемой логикой / /. Работу процессора коротко можно прокомментировать следующим образом.

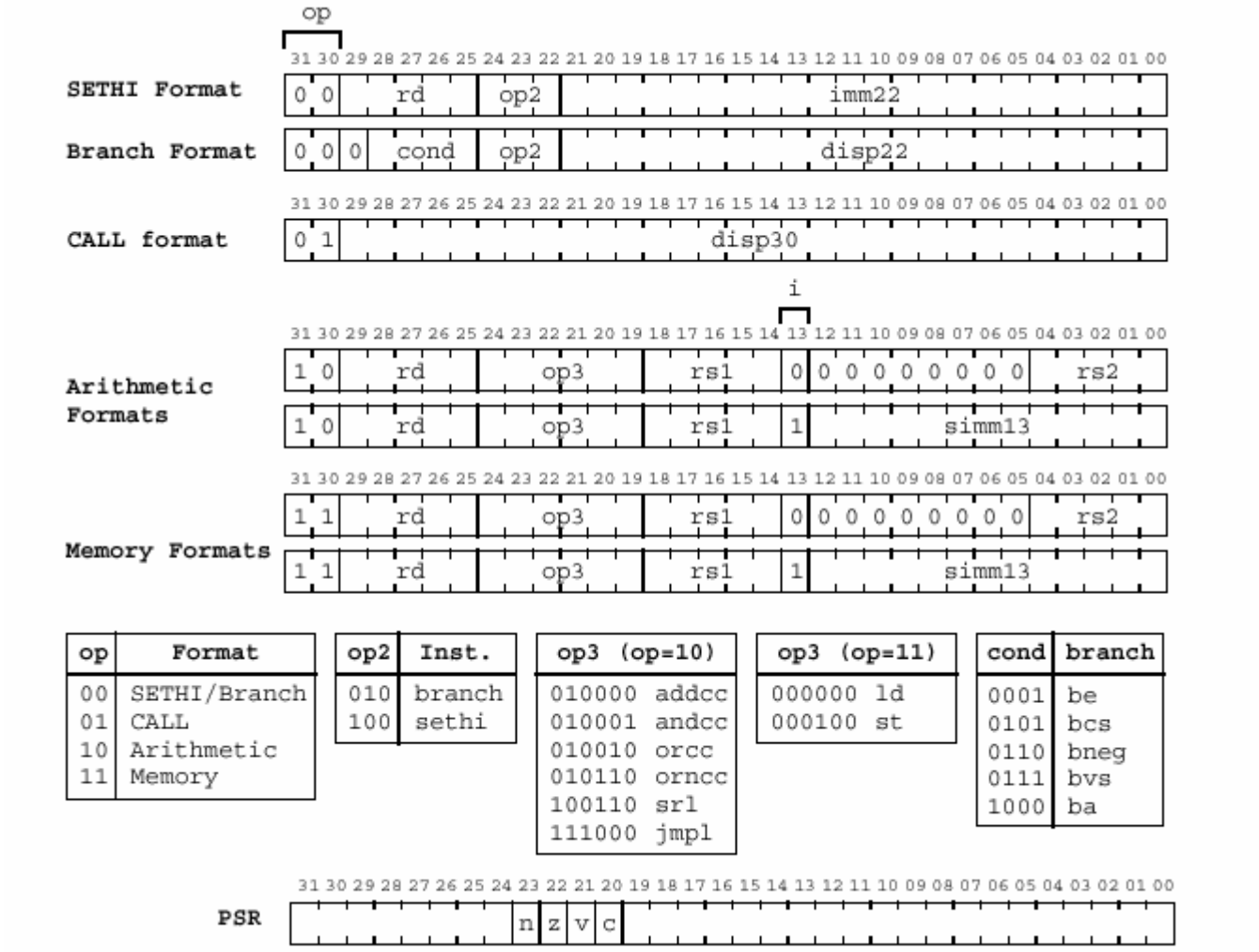
Машинный цикл выполнения команды в общем случае (не для рассматриваемого процессора) включает:

1. Извлечение команды из памяти (IF - Instruction Fetch).
2. Декодирование команды (Instruction Decoding – ID).
3. Извлечение операндов из памяти или из регистров (MEM).
4. Выполнение (Execute - EX).
5. Запись результатов в память или регистр (Write Back – WB).

Для данного процессора обращение к памяти (MEM) и (WB) происходят только в 2 командах – ld и st. В остальных случаях все действия происходят с регистрами РОН. Поскольку у процессора ARC нет отдельного адресного операционного устройства, а режимы адресации предусматривают в том числе и косвенную адресацию, то этап выполнения EX в нем предшествует этапу обращения к памяти (MEM или WB) – на этом этапе необходимо вычислить окончательный адрес памяти, по которому будет обращение.

Машинный цикл выполнения команды реализуется микропрограммно – выполнение каждой команды начинается с нулевого адреса ПМП, где стоит микрокоманда обращения к памяти. После считывания команды идет ее декодирование – код операции в режиме декодирования (Decode) непосредственно используется для формирования адреса следующей микрокоманды – это будет адрес микроподпрограммы для реализации остальных этапов выполнения команды (исполнение и, возможно, обращение к памяти). На этапе исполнения в памяти микропрограмм реализуется собственно алгоритм выполнения каждой команды. Каждая микрокоманда в алгоритме при выполнении помещается в регистр микрокоманд и управляет пересылкой между регистрами, выполнением операций над ними и т.д. Все микрокоманды реализуются через АЛУ (например, пересылка из регистра 1 в регистр 2 может реализовываться как логическая операция «ИЛИ» в АЛУ с нулевым непосредственным значением для регистра 2 и записью результата в регистр 1). После выполнения собственно команд процессора выполняется переход на начало памяти микропрограмм, и весь машинный цикл повторяется для следующей команды (адрес которой находится в регистре PC.)

Таблица 3.2



В результате среднее число тактов на команду (clocks per instruction - CPI) – около 3-4 на команду, и, кроме того, 1 загрузка команды из памяти. В командах обращения к памяти требуется 1 дополнительное обращение к памяти.

Производительность этого процессора можно оценить следующим образом. Среднее время выполнения (в тактах) :

Тк = 3t + 1,5tmem,

где t – длительность одного такта процессора, = tmem – длительность обращения к памяти. При тактовой частоте 100Мгц t=10нс. Пусть время обращения к памяти составляет даже 20нс. Получаем Тк = 3*10нс + 1,5*20нс = 60нс. Производительность = 1/Тк = 1/60нс = менее 20 MIPS. Показатели производительности многих современных процессоров (и RISC и CISC) даже на той же частоте намного выше. (Например, Celeron 400 Мгц имеет производительность около 1000 MIPS – на частоте 100 МГц он бы имел производительность 250MIPS, то есть в 10 раз больше, чем у рассмотренного процессора). Как достигается повышение производительности ? Во-первых, можно несколько улучшить показатель CPI, если перейти к жесткой логике управления, то есть вместо микроподпрограммы выполнения команды реализовать аппаратную схему, выполняющую алгоритм заданной команды.

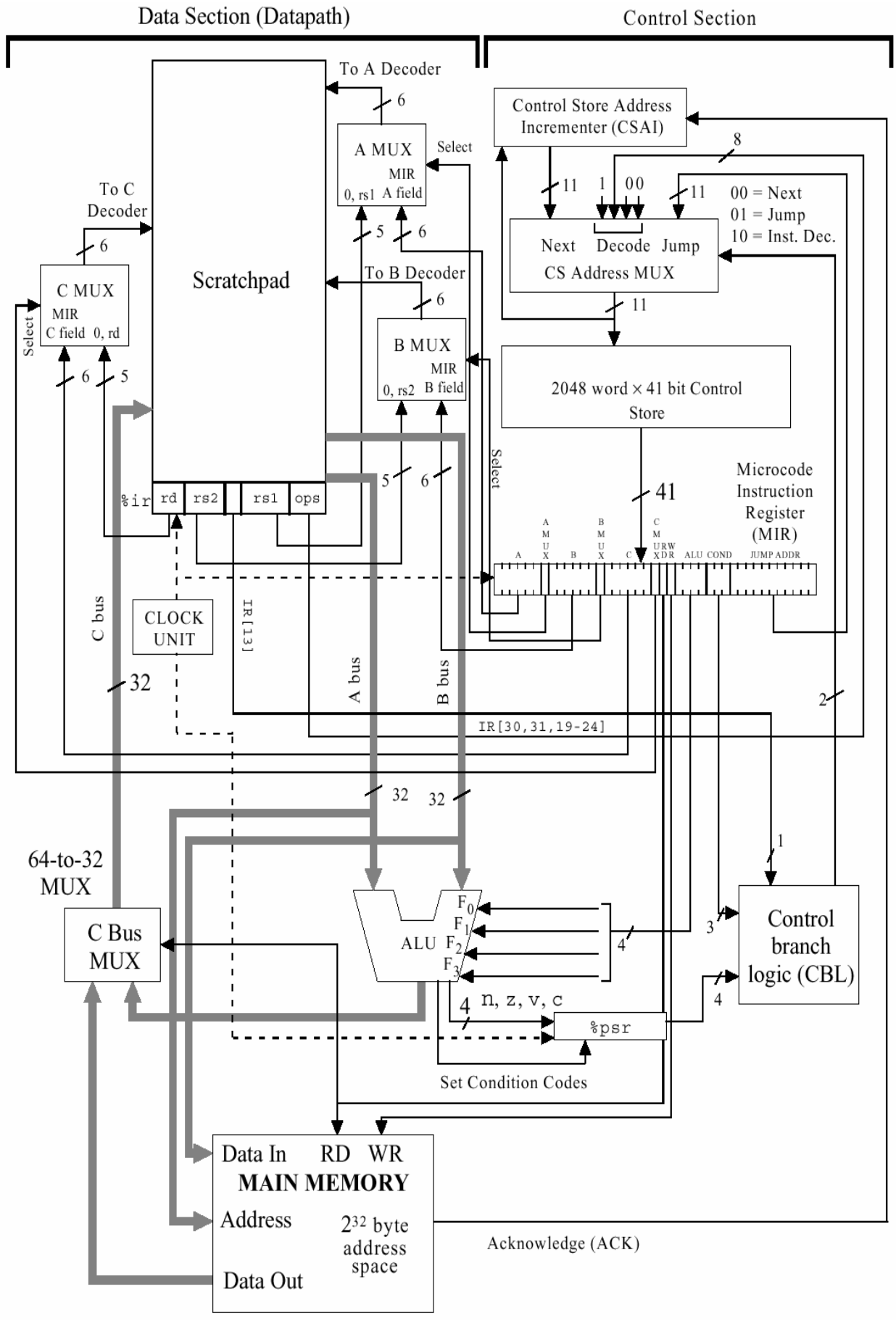

Data Section (Datapath)
Control Section
To A Decoder
6
A MUX
MIR
0, rs1 A field
Select
To C Decoder
6
C MUX
MIR
C field 0, rd
Select
6 5
Scratchpad
To B Decoder
6
B MUX
MIR
0, rs2 B field
5 6
%ir
rd
rs2
rs1
ops
Control Store Address Incrementer (CSAI)
8
11
1
00
11
00 = Next
01 = Jump
10 = Inst. Dec.
Next
Decode
Jump
CS Address MUX
11
2048 word × 41 bit Control Store
41
Select
Microcode Instruction Register (MIR)
A MUX
B MUX
C MUX
A
B
C
URW
DR
ALU
COND
JUMP ADDR
C bus
32
CLOCK UNIT
IR[13]
A bus
B bus
2
IR[30,31,19-24]
32
32
64-to-32 MUX
C Bus MUX
ALU
F0
F1
F2
F3
4
3
1
Control branch logic (CBL)
4
4
n, z, v, c
%psr
Set Condition Codes
Data In
RD
WR
MAIN MEMORY
Address
2^32 byte address space
Data Out
Acknowledge (ACK)

Рис. 3.8

С другой стороны, можно использовать КЭШ-память для ускорения доступа к основной памяти. Однако, этих мер недостаточно для повышения производительности в 10 и более раз.

В современных процессорах для повышения производительности применяют, в том числе, 2 основных подхода: конвейеризацию команд и суперскалярное выполнение команд (многопотоковые конвейеры команд). Эти подходы мы и рассмотрим далее.

3.5.3 Конвейер команд

В общем случае приведенные ранее основные пять этапов выполнения команды процессора общего назначения требуют разного времени, но – сопоставимого. Если добиться (введением фиксаторов и синхронизацией), чтобы каждый этап занимал одинаковое время, можно организовать конвейер команд, в котором одновременно на разных этапах выполнения будут находиться несколько команд (Рис.3.9).

Даже при условии некоторого увеличения времени выполнения одной команды (небольшое снижение быстродействия) производительность при полном заполнении конвейера будет близка к величине $1/T_к$, где $T_к$ – такт конвейера, в данном случае – время выполнения одного этапа. Это позволило бы сразу увеличить производительность процессора в 5 раз ! Однако на практике добиться этого оказывается сложно. И препятствуют этому так называемые конфликты при конвейеризации.

Конфликтом при конвейеризации команд называют ситуацию, которая препятствует выполнению очередной команды из потока команд в предназначенном для нее такте.

Конфликты делятся на три основные группы:

1. Структурные или ресурсные.

Возникают в результате того, что аппаратные средства не могут поддерживать все комбинации команд в режиме их одновременного выполнения с совмещением на конвейере. Это происходит в случае, если какие-то устройства в процессоре не конвейеризованы, либо – присутствуют в единственном экземпляре (не распараллелены). Например, могут возникать конфликты при обращении к общей КЭШ-памяти: одну команду необходимо извлечь из памяти (на первом этапе выполнения), а другая пытается записать результат в память на заключительном этапе. Для борьбы с ресурсными конфликтами в основном применяют три способа:

- приостановка конвейера (pipeline stall, pipeline “bubble” – конвейерный “пузырь”) до разрешения конфликта (до завершения первой конфликтующей команды);
- дублирование аппаратных средств, вызывающих конфликт, например, разделение КЭШ-памяти на КЭШ команд и КЭШ данных;
- ускорение или конвейеризация проблемного устройства, что позволяет снизить затраты времени на приостановку.

Решение об увеличении аппаратных затрат в последних двух случаях принимают, если конфликт возникает часто, так как дополнительные затраты могут быть существеннее, чем потери производительности от приостановки конвейера, если она происходит редко.

2. Конфликты программные или информационные. Делятся на две подгруппы:

а) конфликты по данным, возникающие в случае, если выполнение следующей команды зависит от результата предыдущей.

б) конфликты по управлению, возникающие при нарушении естественного порядка следования команд (условная передача управления).

Выделяют несколько вариантов конфликтов по данным:

1) Конфликт типа «чтение после записи» (Read After Write - RAW). Допустим, имеются две команды – команда Ai и команда Aj, причем команда Ai предшествует команде Aj. Конфликт RAW возникает, если команда Aj использует результаты работы команды Ai, то есть должна прочитать регистр, либо память после записи туда результата командой Ai, но к моменту чтения данные еще не записаны, поскольку команды следуют друг за другом на конвейере и сдвинуты всего на один этап. Например:

ADD R1, R2, R3
SUB R4, R5, R1

IF	ID	MEM	EX	WB	
	IF	ID	MEM	EX	WB

Рис. 3.9.

На рисунке 3.9. обведены этапы, на которых будет записан результат в первой команде и потребуется считать результат для второй команды. Очевидно, что нужный результат еще не будет находиться по месту, адресуемому второй командой, и произойдет конфликт.

2) Конфликт типа «запись после чтения» (WAR). Происходит, если команда Aj записывает результат до того, как он считывается командой Ai (предшествующей Aj). Такой тип конфликта может возникать только в случае, если команда Aj обгоняет команду Ai на конвейере в конвейерах с неупорядоченной обработкой, выдачей или завершением команд (out-of-order execution, out-of-order completition).

3) Конфликт типа «запись после записи» (WAR). Возникает, если последующая команда Aj записывает результат до того, как запишет его команда Ai, что может привести к нарушению логики программы, если, например, между этими командами стоит еще какая-нибудь команда, проверяющая этот адрес (регистр) (пример). Такой конфликт также может происходить в случае неупорядоченного выполнения команд.

Методы борьбы с конфликтами по данным:

а) остановка конвейера;

б) реализация механизмов обхода и продвижения данных (data bypassing & forwarding);

в) планирование загрузки конвейера компилятором (статическая оптимизация);

г) неупорядоченное выполнение команд в процессоре (динамическая оптимизация);

д) переименование регистров.

Рассмотрим перечисленные способы б) – д).

Механизмы обхода и продвижения данных основаны на идее непосредственной передачи результата с выхода одного функционального блока на вход другого, которое в нем нуждается. На рис. 3.10 показано, как результаты могут передаваться непосредственно с выхода АЛУ на вход (через буферные регистры) для того, чтобы обеспечить следующие команды самыми «свежими» данными предыдущей операции, минуя этапы записи в память или регистр.

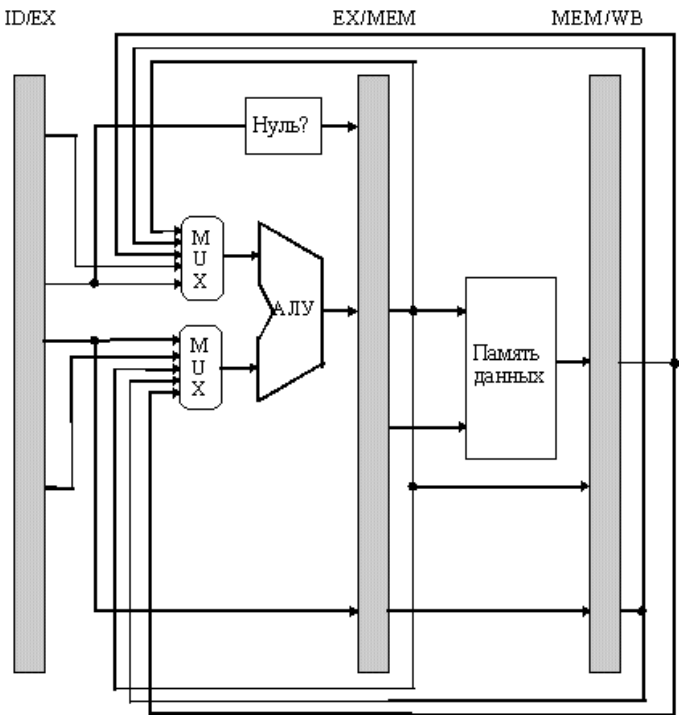


Рис. 3.10

Статическое планирование позволяет еще на этапе компиляции переупорядочивать команды таким образом, чтобы они по возможности не конфликтовали друг с другом. Например, рассмотрим 2 оператора языка :

A := B + C;
D := E + F.

Неоптимизированный код, реализующий эти операции, представлен в первой колонке таблицы 3.1. В данном потоке команд возникают конфликты типа RAW после второй и седьмой команд, а также после 4-ой и восьмой. Конфликты при записи в память могут быть устранены с помощью механизма обходов, а конфликты при чтении памяти – с помощью переупорядочения потока команд, как показано во второй колонке таблицы 3.3.

При динамической оптимизации команды, вызвавшие конфликт, могут задерживаться на конвейере, а следующие за ними команды, не вызывающие конфликтов и не зависящие от конфликтных команд, пускаются в обход них. Анализ конфликтных ситуаций и выбор команд для переупорядочивания возлагается на логику управления конвейером в самом процессоре.

Таблица 3.3.

Исходная последовательность команд	Переупорядоченная последовательность
LW Rb, B	LW Rb, B
LW Rc, C	LW Rc, C
ADD Ra, Rb, Rc	LW Re, E
SW A, Ra	ADD Ra, Rb, Rc
LW Re, E	LW Rf, F
LW Rf, F	SW A, Ra
ADD Rd, Re, Rf	ADD Rd, Re, Rf
SW D, Rd	SW D, Rd

При переименовании регистров логические имена регистров, присутствующие в командах, динамически отображаются на физические регистры процессора, которых, как правило, больше (если речь идет о RISC-подобной архитектуре). В результате различные данные, относящиеся к одному и тому же логическому регистру, помещаются в разные физические регистры. Соответствие между регистрами задается таблицей отображения, которая динамически обновляется после декодирования каждой команды. Это облегчает процессору планирование загрузки конвейера.

Конфликты по управлению возникают при конвейеризации команд условных переходов. По статистике до 15%-20% всех команд в программе - это команды условных переходов. Конфликт проявляется в том, что адрес условного перехода определяется только в конце выполнения команды, в то время, как конвейер уже должен быть заполнен командами из какой-то одной ветви. Если не обрабатывать конфликт, то производительность конвейера может снижаться в 2 и большее число раз. Способы борьбы с конфликтами по управлению можно разделить на 2 группы: статические и динамические.

К статическим методам можно отнести:

1) Возврат, т.е. статическое прогнозирование перехода как всегда выполняемого, либо – всегда не выполняемого с последующей очисткой конвейера в случае неправильного прогноза и возвратом к нужной команде.

2) Разворачивание циклов. Поскольку многие условные переходы связаны с определением условия выхода из цикла, то в случае, когда число шагов цикла заранее известно и невелико, можно «развернуть» цикл, то есть продублировать тело цикла столько раз, сколько необходимо, отказавшись от проверки условия вообще. Это приведет к увеличению кода программы, но избавит от конфликта по управлению.

3) Задержанные переходы (использование «слотов задержки»).

Под слотами задержки понимают участки программы, которые будут занесены в конвейер и успеют выполниться до выяснения адреса перехода в условной команде. Идея их использования состоит в том, чтобы заполнять слоты задержки «полезными» или «невредными» командами, то есть такими, которые

либо не зависят от условия, либо – не приведут к нарушению логики программы, даже если их выполнение окажется лишним. (Рис. 3.11)

4) Предсказание переходов на основе профиля программы. Профилирование (profiling) предусматривает составление прогноза о выполнении тех или иных переходов на основе статистических наблюдений за программой по результатам ее многократного прогона.

К динамическим методам можно отнести:

1) Приостановку конвейера до выяснения адреса перехода.
2) Реализацию команд условного перехода в процессоре таким образом, чтобы адрес перехода выяснялся на начальных этапах выполнения команды.
3) Динамическое предсказание ветвлений в процессоре (branching predict).

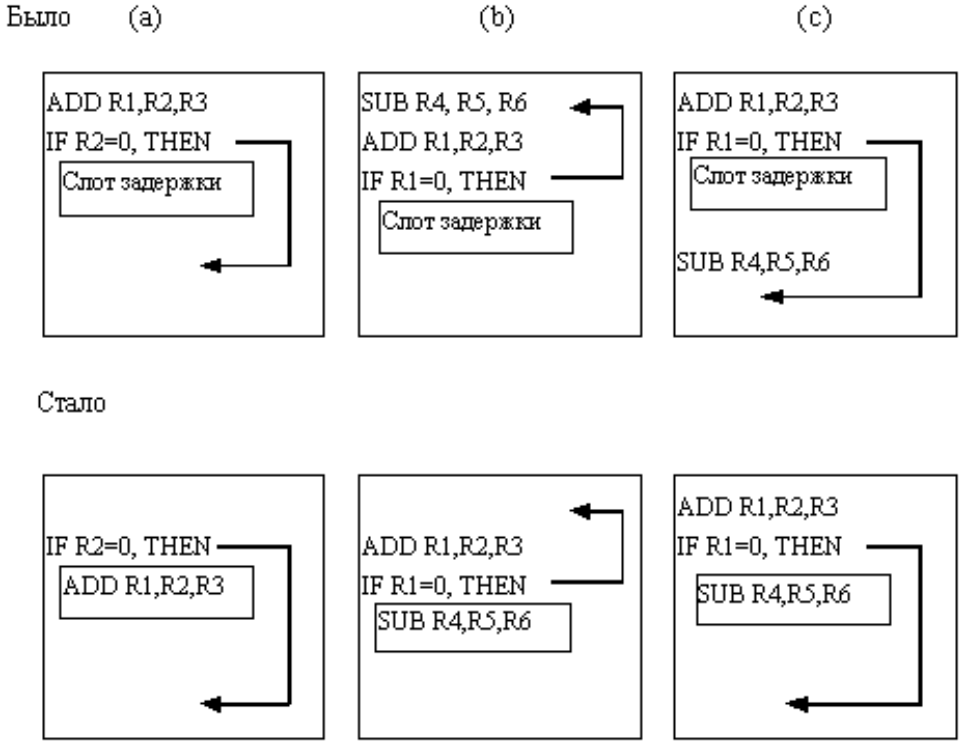


Рис. 3.11

Динамическое предсказание ветвлений в процессорах осуществляется с помощью буферов предсказания перехода (БПП – Branch Predicting Buffer – BPB). Чаще всего в них используется счетчик прогнозов, который представляет собой обычный n-разрядный двоичный счетчик. При каждом выполненном переходе счетчик прогнозов для данного перехода увеличивается, а при невыполненном – уменьшается на единицу. Если текущее значение счетчика > 2^{n-1}, то переход прогнозируется как выполняемый, иначе – как невыполняемый. На практике ограничиваются либо 1-битным, либо 2-битными счетчиками, которые при этом обеспечивают вероятность правильного прогноза соответственно до 70% и 85%.

Для еще большего ускорения предсказания используют буфер целевых адресов переходов (Branch Target Buffer – BTB), представляющий собой

ассоциативную кэш –память, в которой в качестве тегов используются адреса команд ветвления в текущей части программы, а в ячейках содержатся счетчики прогнозов и целевые адреса перехода при условии его выполнения. Процессор при выборке команды проверяет, не хранится ли ее адрес в BTB, считывает счетчик прогнозов и в зависимости от его значения принимает решение о выборке команд по следующему адресу или по адресу, указанному в BTB.

3.5.4 Суперскалярные архитектуры

Итак, использование конвейера команд позволяет в лучшем случае снизить показатель CPI до 1, то есть на каждом такте с конвейера должна «сходить» новая обработанная команда. В этом случае производительность нашего процессора ARC должна увеличиться в 4 раза, при его длительности такта в 10 нс (тактовая частота 100 Мгц) имеем производительность в 100 MIPS. Но во-первых, у Celeron такой показатель равняется, как мы выяснили, где-то 250, а во-вторых – как показано ранее, достижение показателя 1 CPI не всегда возможно из-за конфликтов при конвейеризации. То есть реально мы будем иметь в лучшем случае 1,5-2 CPI. Как же достигается такая высокая производительность в Celeron и других процессорах с архитектурой P6? Для этого в них используется суперскалярная обработка, то есть обработка с многопотоковым конвейером команд, когда процессор может выполнять больше 1 команды за такт (CPI < 1, или – IPC > 1).

Фактически в суперскалярном процессоре несколько потоков проходят через несколько исполнительных устройств, а остальные ступени так или иначе работают с одним потоком. Для согласования разных скоростей потоков декодирования, выборки, трансляции в RISC - подобные инструкции, переупорядочивания и потоков в исполнительных устройствах применяют различные очереди инструкций (буферы FIFO), которые есть у каждого из исполнительных устройств (рис. 3.12)

Необходимо отметить, что в суперскалярных процессорах происходит очень сложное и нетривиальное преобразование последовательного потока команд исходной программы в параллельный поток триад (операция + операнды + назначение результата), параллельно продвигающихся по очередям команд в исполнительные устройства. Процессор ограничен в возможностях такого преобразования, а также в возможностях спекулятивного исполнения (подготовки загрузки альтернативных ветвей ветвления) и прогнозирования ветвлений размером т.н. «окна исполнения», то есть той частью программного кода, который процессор «видит» в процессе выполнения в данном такте. Все это ограничивает возможности распараллеливания потоков команд до величин порядка 6-8 (что тоже в принципе неплохо).

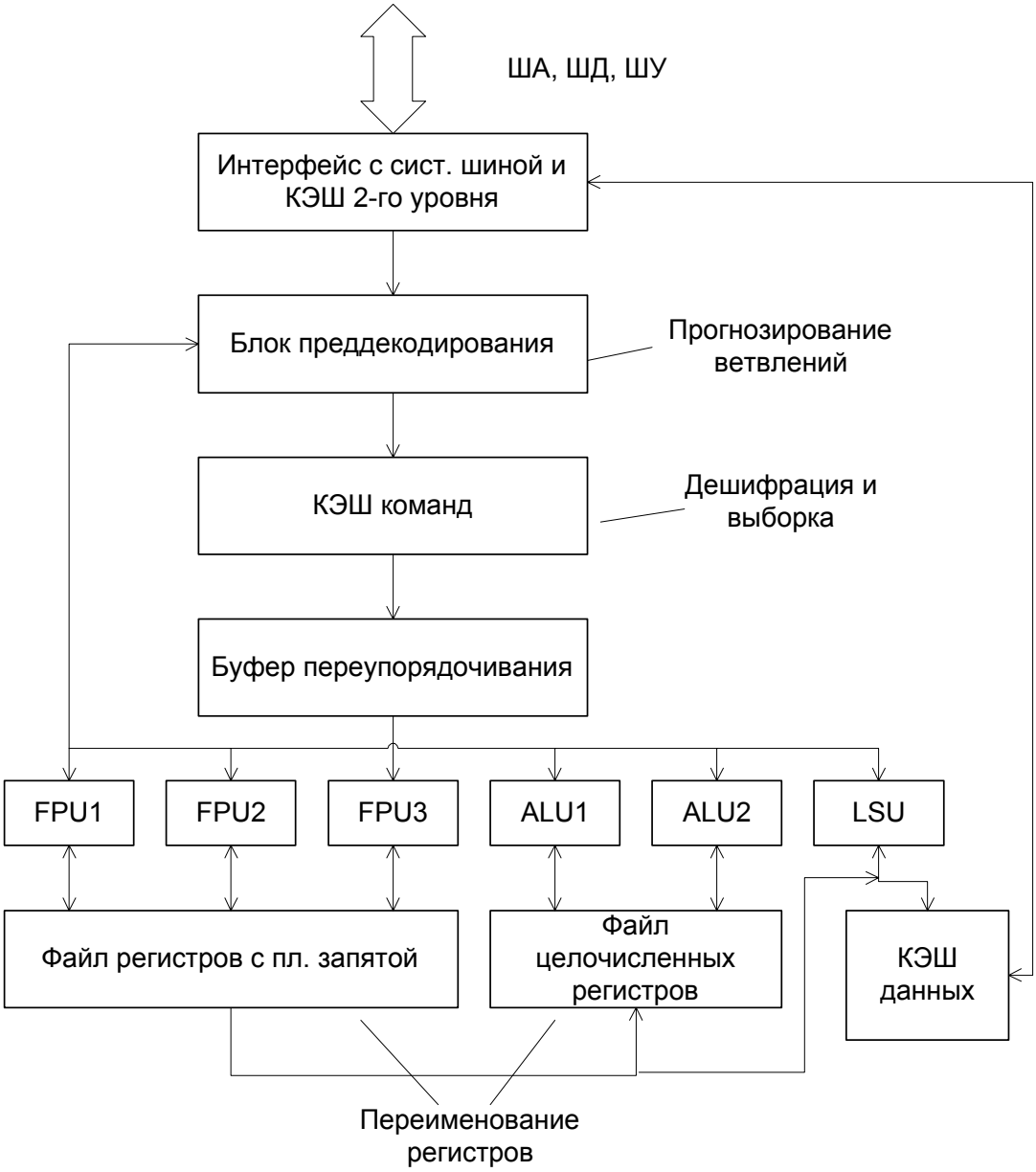


Рис. 3.12

3.5.5 Процессоры с длинным командным словом (VLIW).

В процессорах с длинным командным словом (Very Long Instruction Word) используется альтернативный суперскалярной обработке принцип распараллеливания последовательного алгоритма. В основном вся тяжесть планирования загрузки большого числа исполнительных устройств в таком процессоре (а у него блочное операционное устройство) ложится на программиста, или – на оптимизирующий компилятор. В процессор поступают уже сформированные триады для всех исполнительных устройств, так что ему только остается выполнять эти длинные команды. В результате он не ограничен

размером окна исполнения, так как и программист, и компилятор видят весь код программы, и могут извлечь из него максимальный параллелизм.

Такой подход позволяет достичь принципиально более высокой производительности (например, тестирование процессоров Itanium с архитектурой IA-64, использующей принципы VLIW, указывает на 10-кратное ускорение при выполнении ряда вычислений), но такие процессоры обладают и рядом недостатков:

- в целом менее эффективная загрузка исполнительных устройств, так как не всегда можно сформировать достаточное количество команд для параллельного исполнения;
- сложности обработки условных переходов;
- сложность программирования и др.

Последнее обстоятельство ограничивает применение процессоров VLIW, даже Intel, в персональных ЭВМ, так как для этого придется кардинально переписывать все программное обеспечение, поскольку в существующем виде оно не даст прироста производительности на этих процессорах. Сфера применения VLIW-процессоров пока ограничена серверами, производительными рабочими станциями, а также многопроцессорными ЭВМ.

Что касается обработки условных переходов, то тут можно отметить широкое использование в процессорах VLIW так называемых условных (conditional) команд. Это команды, использующие предварительно рассчитанные логические значения (предикаты), для выполнения, либо пропуска какого-то действия (наподобие операторов языков высокого уровня c := iif (a>b, a, c)), что позволяет избавиться от нескольких ветвей при коротких условных переходах и использовать один поток команд без необходимости предсказывать адрес для следующей выборки.

3.6 Обзор архитектур процессоров Intel

Корпорация Intel является “законодателем мод” на рынке микропроцессоров, а ее продукты стали де-факто стандартом в компьютерной индустрии. Конечно, существует большое количество других производителей и распространенных семейств МП (те же процессоры Motorola и др.), однако наиболее распространенными во всем мире, и особенно в России, являются все же процессоры Intel.

Кроме того, в течении многих лет другие разработчики МП (AMD, NextGen, VIA и др.) выпускают свои аналоги процессоров, совместимых по системам команд с МП Intel. Поэтому, анализируя эволюцию процессоров Intel, мы можем в какой-то степени проследить историю развития микропроцессоров общего назначения вообще.

А история развития процессоров Intel подтверждает в целом закон Мура, сформулированный одним из основателей империи Intel Гордоном Муром еще в 1965 году: “каждые 1,5-2 года выпускается новый процессор, степень интеграции которого (и производительность) вдвое выше, чем у предыдущего.”

Таблица 3.4

Характеристики процессоров Intel (i8086-Pentium III)

Intel Processor	Date Intro-duced	Max. Clock Frequency at Intro-duction	Transis-tors per Die	Register Sizes[1]	Ext. Data Bus Size[2]	Max. Extern. Addr. Space	Caches
8086	1978	8 MHz	29 K	16 GP	16	1 MB	None
Intel 286	1982	12.5 MHz	134 K	16 GP	16	16 MB	Note 3
Intel386 DX Processor	1985	20 MHz	275 K	32 GP	32	4 GB	Note 3
Intel486 DX Processor	1989	25 MHz	1.2 M	32 GP 80 FPU	32	4 GB	L1: 8KB
Pentium Processor	1993	60 MHz	3.1 M	32 GP 80 FPU	64	4 GB	L1:16KB
Pentium Pro Processor	1995	200 MHz	5.5 M	32 GP 80 FPU	64	64 GB	L1: 16KB L2: 256KB or 512KB
Pentium II Processor	1997	266 MHz	7 M	32 GP 80 FPU 64 MMX	64	64 GB	L1: 32KB L2: 256KB or 512KB
Pentium III Processor	1999	500 MHz	8.2 M	32 GP 80 FPU 64 MMX 128 XMM	64	64 GB	L1: 32KB L2: 512KB

Согласно информации Intel /11/, за 24 года количество транзисторов в кристалле МП увеличилось более чем в 3700 раз от 29 тыс. в процессоре i8086 (выпущен в 1978 г.) до 108 млн в Intel Pentium IV. При этом производительность процессоров возросла более чем в 6000 раз (от 0.8MIPS для i8086 до приблизительно 5000 MIPS для Pentium IV 2,6 ГГц) !

Наряду с прогрессом интегральной технологии в ходе эволюции процессоры Intel претерпевали и значительные архитектурные изменения. Если говорить об архитектуре, известной как x86, то она ведет начало от процессора i8086 до наших дней (Pentium III и IV). Существенной особенностью всех процессоров x86 является их совместимость снизу вверх, что позволяет до сих пор пользоваться программами, написанными 20 лет назад! В таблице 3.4 приведены характеристики основных процессоров x86 вплоть до Pentium III, а в таблице 3.5 – характеристики последнего поколения процессоров Pentium IV. Проследим кратко историю эволюции этих процессоров.

Первый процессор семейства x86 (или – Архитектуры Intel) - i8086 - был 16-разрядным, имел 16-разрядную внешнюю шину данных и 20-разрядную шину адреса, что позволяло адресовать до 1Мб внешней памяти. Память имела сегментную организацию с сегментами до 64К. Аналогичный процессор i8088 имел внешнюю шину в 8 бит, что удешевило популярные персональные системы IBM PC/XT. В процессоре i80286 был реализован “защищенный режим” работы, позволявший адресовать до 16Мб памяти, использовавший дескрипторные

таблицы, систему с кольцами защиты памяти и аппаратной поддержкой переключения задач. Это новшество позволило перенести в среду персональных ЭВМ элементы операционных систем больших ЭВМ и майнфреймов – многозадачность, защиту памяти и системных ресурсов.

Таблица 3.5

Характеристики процессоров Intel (P6 – Pentium - IV)

Intel Processor	Date Intro-duced	Micro-Architecture	Clock Frequency at Intro-duction	Transis-tors Per Die	Register Sizes[1]	System Bus Band-width	Max. Extern. Addr. Space	On-Die Caches[2]
Pentium III and Pentium III Xeon Processors[3]	1999	P6	700 MHz	28 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	Up to 1.06 GB/s	64 GB	3-KB L1; 256-KB L2
Pentium 4 Processor	2000	Intel NetBurst Micro-architecture	1.50 GHz	42 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	3.2 GB/s	64 GB	12K µop Execution Trace Cache; 8KB L1; 256-KB L2
Intel Xeon Processor	2001	Intel NetBurst Micro-architecture	1.70 GHz	42 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	3.2 GB/s	64 GB	12K µop Trace Cache; 8-KB L1; 256-KB L2
Intel Xeon Processor[4]	2002	Intel NetBurst Micro-architecture; Hyper-Threading Technology	2.20 GHz	55 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	3.2 GB/s	64 GB	12K µop Trace Cache; 8-KB L1; 512-KB L2
Intel® Xeon™ Processor MP[4]	2002	Intel NetBurst Micro-architecture; Hyper-Threading Technology	1.60 GHz	108 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	3.2 GB/s	64 GB	12K µop Trace Cache; 8-KB L1; 256-KB L2; 1-MB L3

Процессор i80386 был 32-разрядным, адресовал до 4Гб памяти, включал аппаратную поддержку виртуальной памяти и возможность адресовать всю память в «плоском» режиме. На машинах с этим процессором можно было реализовывать операционную систему UNIX – классическую систему майнфреймов. В процессоре 386 уже использовалось распараллеливание при одновременной работе 6 основных устройств процессора.

В процессоре i80486DX появилась встроенная КЭШ-память 1 уровня (L1) объемом 8К, встроенный математический сопроцессор, а также – пятиступенчатый конвейер в устройствах декодирования и исполнения команд.

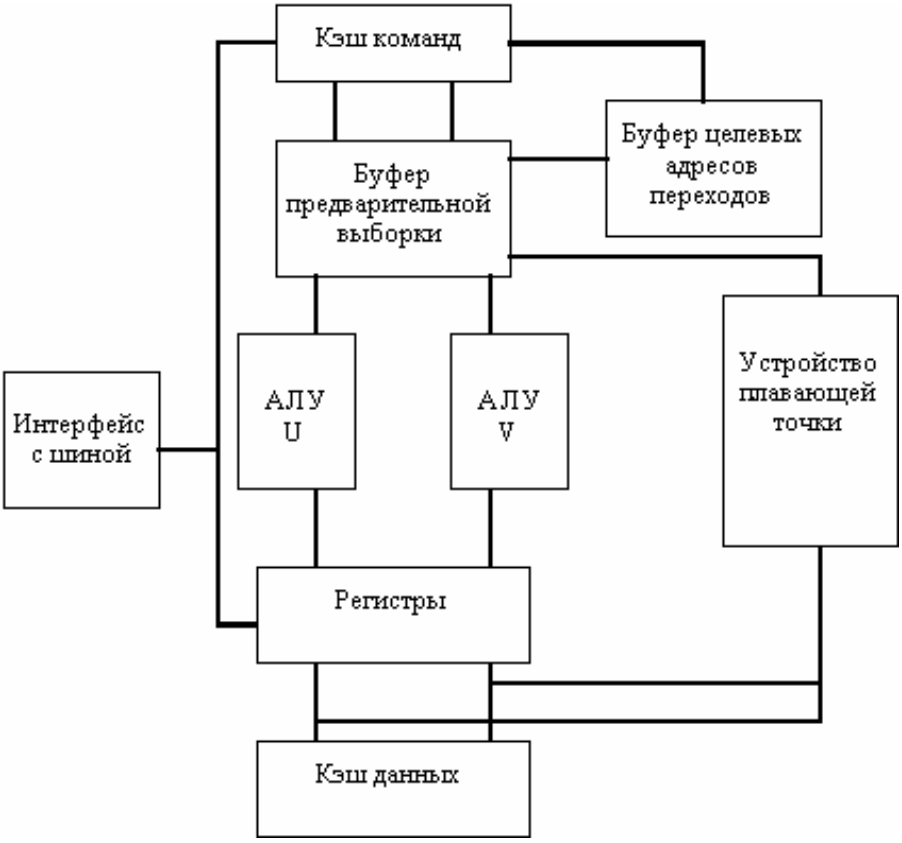


Рис. 3.13

Процессор Intel Pentium явился первым суперскалярным процессором (Рис.3.13). В нем был реализован двухпотоковый конвейер, который позволял одновременно обрабатывать до двух команд в такте. (Правда, при этом существовало много ограничений, но тем не менее.)

Процессор включал раздельную КЭШ-память для инструкций и данных (по 8К), поддерживающую режим обратной записи. Внутренняя разрядность процессора осталась 32 разряда, но некоторые внутренние шины имели разрядность 128 и даже 256 разрядов, а внешняя шина – 64 разряда. В процессоре было реализовано динамическое предсказание переходов и поддержка мультипроцессорных конфигураций.

Появление процессора Pentium Pro дало начало новой модификации Intel Architecture – архитектуре P6. Процессор (рис. 3.14) имеет 3-х потоковый конвейер, что позволяет достичь большей степени распараллеливания по сравнению с обычным Pentium. Главной отличительной особенностью процессора является, пожалуй, динамическое исполнение (Dynamic Execution) – реализация неупорядоченного выполнения, спекулятивного исполнения (исполнения по предположению) и усовершенствованного блока предсказаний. В процессоре реализована суперконвейерная архитектура, поскольку он содержит 13 более мелких ступеней конвейера по сравнению с 5 у Pentium, на которых исполняются специальные RISC-подобные инструкции процессора, получившие названия micro-ops. Три буфера декодирования параллельно формируют три потока таких инструкций, которые затем направляются в пять исполнительных устройств, результаты обработки в которых затем собираются в правильном порядке в блоке сборки.

Процессор содержит две КЭШ-памяти L1 по 8К, а также – КЭШ второго уровня (L2) объемом в 256К, реализованную в том же корпусе, что и основной кристалл процессора и обменивающуюся с ним по скоростной шине шириной в

64 разряда. Шина адреса процессора имеет 36 разрядов, что обеспечивает адресное пространство в 64 Гбайт.

Как мы видим, прогресс в области архитектур процессоров Intel во многом определялся последовательной реализацией способов борьбы с конфликтами при конвейеризации, рассмотренными нами ранее.

Уже после реализации Pentium Pro Архитектура Intel дополнилась реализацией векторных SIMD – инструкций, которая получила название технологии MMX (MultiMedia eXtensions – мультимедийные расширения). Суть новой технологии заключалась в реализации с помощью широких регистров математического сопроцессора целочисленных команд обработки векторов размерностью до 8 чисел по 8 бит. Эти команды позволяли ускорить обработку видео и аудио информации за счет ускорения векторных и матричных операций. Так появились процессоры Pentium MMX, с максимальной частотой до 233Мгц (300 Мгц в варианте для notebook).

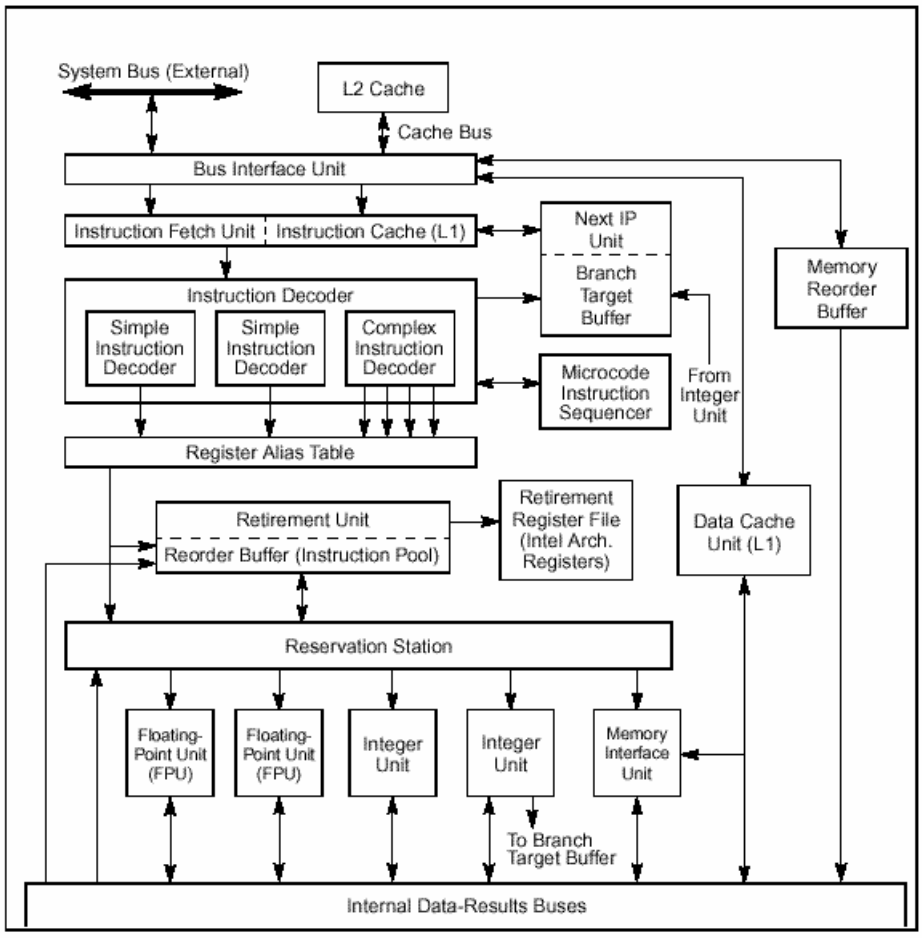


Рис. 3.14

Процессор Pentium II стал дальнейшим развитием архитектуры P6 (фирма Intel широко использует для своей основной архитектуры обозначение IA32). Архитектурно процессор стал симбиозом Pentium Pro с поддержкой MMX – инструкций. Кроме того, в нем реализована КЭШ-память с более быстродействующей шиной и большего объема, а также ряд технологических и

конструктивных улучшений, связанных с выбором корпуса, процессорной шины и т.д.

Процессор Pentium III, заявленный как процессор, специально спроектированный для поддержки Internet-приложений, в дополнение к стандартному набору инструкций MMX включает набор из 70 новых инструкций SSE (Streaming SIMD Extensions – потоковые расширения SIMD), которые позволяют расширить векторную обработку за счет обработки чисел с плавающей запятой и реализации 3D-инструкций в универсальном процессоре. Кроме того, теперь каждый процессор снабжен уникальным серийным номером для его идентификации в сети Internet.

Ряд серьезных изменений произошел в архитектуре процессора Pentium IV. Этот процессор уже не относится к архитектуре P6, хотя продолжает линейку IA-32. К его ключевым особенностям можно отнести кэш второго уровня, помещенный на кристалл процессора, увеличенную до 400(533)МГц частоту передней шины, 144 новых векторных инструкций SSE2, хранение так называемых трасс декодированных внутренних инструкций в специальном КЭШе и другие изменения.

4. ОРГАНИЗАЦИЯ ВВОДА-ВЫВОДА

4.1 Назначение и состав системы ввода-вывода. Варианты обмена в СВВ

Система ввода-вывода (СВВ) предназначена для выполнения трех из пяти основных функций ВМ: собственно ввода, вывода и долговременного хранения информации (две другие функции – это обработка информации и управление этой обработкой). Соответственно, в состав системы ввода-вывода входят:

1. Периферийные устройства, которые можно подразделить на:
   - устройства ввода информации;
   - устройства вывода (отображения) информации;
   - устройства вывода информации на исполнительные устройства (управление техническими и производственными объектами и т.д.);
   - устройства долговременного хранения информации.
2. Контроллеры (адаптеры) периферийных устройств, отвечающие за управление периферийными устройствами и обмен между ними и ядром ЭВМ.
3. Внешние параллельные и последовательные каналы обмена и соответствующие контроллеры этих каналов.
4. Системные шины и их контроллеры.
5. Контроллеры прерываний и прямого доступа к памяти (ПДП).
6. Подсистемы процессора, отвечающие за ввод/вывод, организацию ПДП и реакцию на прерывания и др.

Традиционно выделяют три основных варианта обмена процессора с внешними устройствами:

- программный обмен (program polling);
- обмен по прерываниям (interrupts);
- обмен в режиме прямого доступа к памяти – ПДП (direct memory access – DMA).

При программном обмене процессор постоянно опрашивает внешнее устройство, дожидаясь его готовности к обмену или наступления другого события. Такой вариант является самым простым, так как требует только чтения портов ввода/вывода, определенные биты которых сигнализируют о наступлении ожидаемого события, однако он приводит к непродуктивным потерям времени процессором, даже если он выполняет свои запросы не постоянно, а через определенные (длительные) интервалы времени. С другой стороны, при таком варианте, если длительность интервалов достаточно велика, процессор может и пропустить какое-то событие.

При обмене по прерываниям процессор не ждет наступления события, а реагирует на него по факту его наступления, когда формируется сигнал прерывания (подробнее – дальше). В результате процессор сможет отреагировать на все события, без пропуска, и при этом освобождается от необходимости постоянно проводить опрос внешних устройств. Традиционно обмен по прерываниям предназначался в основном для «медленных» внешних устройств, но может использоваться и для быстрых, для инициирования начала обмена

(который происходит программно), либо – для отслеживания процессором внешних событий без постоянных запросов.

При обмене в режиме ПДП процессор освобождается от необходимости выполнять «черновую» работу по пересылке информации из внешнего устройства в память и наоборот через свои регистры. Раньше, когда быстродействие процессоров было невелико, режим ПДП позволял ускорить обмен при исключении медленного посредника, каким являлся процессор. В настоящее время назначение ПДП по большей части в другом – разгрузить процессор для выполнения более насущных задач.

4.2. Система прерываний

Рассмотрим немного подробнее характеристики прерываний и их реализацию.

Прерывание – это реакция ЭВМ на некоторое внутреннее или внешнее событие, приводящая к приостановке выполнения текущей программы и инициированию выполнения новой программы, специально предназначенной для данного события.

Основные типы прерываний:

1. Внешнее прерывание, инициируемое внешним устройством. Внешние прерывания могут быть маскируемыми (т.е. такими, на которые может накладываться маска для исключения прерывания из обработки) и немаскируемыми, на которые процессор должен реагировать в любом случае.
2. Программные прерывания, вызываемые самой программой. Служат для обращения к системным сервисам операционной системы или BIOS для выполнения низкоуровневых задач, детали выполнения которых могут быть скрыты от программы (например – вывод символа на экран). Фактически эти прерывания являются вызовом подпрограмм, но – заранее подготовленных и не являющихся частью основной программы.
3. Внутренние прерывания (исключения – exceptions), генерируемые процессором как реакция на ошибки, например, на попытку деления на 0 или обращения программы к запрещенной для нее области памяти. Исключения помогают обработать ошибку (средствами самой программы или операционной системы).

Основные этапы выполнения прерываний :

1. Запоминание состояния прерываемой программы.
2. Определение адреса подпрограммы-обработчика прерывания (вектора прерывания).
3. Переход к выполнению подпрограммы обработки прерывания.
4. Восстановление состояния прерванной программы и возврат к ней.

Важной задачей системы прерываний является определение вектора прерываний, для чего необходимо распознать источник прерываний. Это выполняется в самом процессоре, либо – с помощью специального устройства – контроллера прерываний. На контроллер прерываний может также возагаться

задача управления приоритетами в соответствии с заданными программой установками.

При сохранении состояния прерываемой программы необходимо сохранить содержимое регистров процессора, состояние флагов, конвейеров процессора, состояние системы защиты памяти, различную служебную информацию и т.д.

Сохранение состояния прерываемой программы чаще всего происходит в стеке системы, поэтому зачастую количество прерывающих друг друга прерываний ограничено размером свободной части стека.

Характеристики системы прерываний:

1. Общее число возможных запросов.
2. Число уровней или классов прерываний.
3. Глубина прерываний – максимальное количество запросов, которые могут прерывать друг друга.
4. Время реакции системы прерываний – время от формирования запроса до начала выполнения подпрограммы обслуживания прерывания.
5. Эффективность системы прерываний – отношение времени выполнения п/п обработчика прерывания к общему времени, затрачиваемому на прерывание (складывается из времени реакции, времени выполнения п/п обработчика и времени восстановления).
6. Используемая система приоритетов (количество маскируемых и немаскируемых прерываний, способ изменения приоритетов и т.д.)

4.3. Шинно-мостовая организация системы ввода-вывода. Системные шины

Под шиной (bus) будем понимать в общем случае многоразрядный интерфейс для обмена информацией. Как правило, такой интерфейс так или иначе стандартизован.

Описание интерфейса включает:

- логическое описание (количество и назначение используемых разрядов);
- физическое описание (количество разрядов, способ физического сопряжения с интерфейсом);
- электрическое описание (используемые уровни сигналов и др.);
- протокольное описание (похоже на логическое, но описывает динамику выставления, снятия и взаимозависимости сигналов интерфейса).

Мост (bridge) – в данном случае устройство для согласования двух различных интерфейсов.

Шинная архитектура широко используется в ЭВМ при построении систем ввода/вывода, да и ядра вычислительной системы. Как правило, шинный интерфейс включает линии данных, адреса и управления, может также включать линии питания, «логическую землю» и т.д.

Под системной шиной можно понимать шину, служащую для связи различных компонентов вычислительной системы в единое целое. При этом существует понятие локальной системной шины, подразумевающее, что действие и назначение шины в рамках системы достаточно локально, но расположена

шина внутри системы, например, возле ядра. В противоположность системным внешние шины служат для связи ЭВМ с внешними устройствами. Они являются в общем случае более стандартизированными, для того, чтобы внешние устройства различных производителей могли сопрягаться с различными ЭВМ.

Исторически в ПК первой системной шиной можно считать шину ISA (Industrial Standard Architecture – индустриальный (промышленный) стандарт архитектуры). Она включает 16-битную (первоначально – даже 8-битную) шину данных и работает на частоте до 8,33 МГц. Модификациями этой шины можно считать шины EISA (Extended ISA), Microchannel, VLB (VESA Local Bus – локальная шина VESA, где VESA – Video Equipment Standarts Association - ассоциация стандартов по видеооборудованию).

Шина ISA в том или ином виде стала основой для выделения более специализированных шин, таких, как IDE (Integrated Device Electronics – “интегрированная в устройстве электроника») – шина для подключения дисковых накопителей. Стандарт IDE развился в шину EIDE, а затем – в стандарт ATA (AT attachment – «подключение к AT») и ATAPI (протокол подключения к ATA) – стандарт для подключения разнородных накопителей, таких, как HDDm CD-ROM, DVD.

Современные дисковые подсистемы включают поддержку стандарта ATA33/66/100. Стандарт ATA33 – UlltraDMA33 позволяет повысить скорость обмена до 33Мб/с за счет синхронизации передач как по переднему, так и по заднему фронтам тактовых импульсов. В стандарте ATA66 – UltraDMA66 предполагается увеличение частоты шины до 16МГц, а также – возможность увеличения разрядности шины до 32 разрядов.

Современные ПК (и не только) строятся на базе шины PCI – Peripheral Component Interconnect, активно поддерживаемой корпорацией Intel. Шина PCI стандартом системной шины не только для ПК различных архитетктур, но и для систем промышленного назначения, больших ЭВМ и т.д.

Особенности шины PCI:

- шина является универсальной, то есть, не привязана к конкретной архитектуре ЭВМ;
- разрядность ШД – 32 бита, используется совмещение ША и ШД (разделение передач во времени);
- тактовая частота – 33 МГц (существуют варианты PCI с 64МГц и даже – 166 Мгц);
- шина использует достаточно быстродействующий протокол;
- включает поддержку мостов и др.

Наличие поддержки мостов позволяет строить иерархические шинные архитектуры на базе PCI.

На современном этапе шинно-мостовая архитектура в ПК организована чаще всего по принципу “Север-Юг”, с двумя основными мостами – северным (Northbridge) и южным (Southbridge). В совокупности эти два моста и необходимое окружение образуют «чипсет» системной платы компьютера.

Функцией северного моста является обеспечение взаимодействия ЦП, оперативной памяти и видеоподсистемы (как самого быстродействующего

интерфейсного модуля), а также - интерфейс ядра ВС с системной шиной PCI. Южный мост включает контроллеры для связи с внешними устройствами по интерфейсам ISA (мост PCI-ISA), EIDE, USB, FireWire, контроллеры параллельных и последовательных портов, клавиатуры, портов PS/2 и др. Северный мост также может обеспечивать связь с конрроллером SCSI и другими устройствами. Между северным и южным мостами, связанными между собой шиной PCI, располагаются физические интерфейсы для подключения внешних плат PCI. Помимо основного южного моста, расположенного на плате, к северному мосту через шину PCI могут подключаться дополнительные внешние южные мосты, со своими наборами внешних интерфейсов, что делает такую архитектуру достаточно открытой. Преимуществом разделения «Север-Юг» является возможность отделить специфичную и высокоскоростную архитектуру ядра (оптимизированную под конкретные процессоры, модули памяти и видеоподсистему) от стандартной системной шины и внешних интерфейсов (Заметим, что частота «передней» шины (Front Side Bus) в современных системах может достигать 150-166 МГц, а для систем на базе Pentium IV – 400 Мгц !!!).

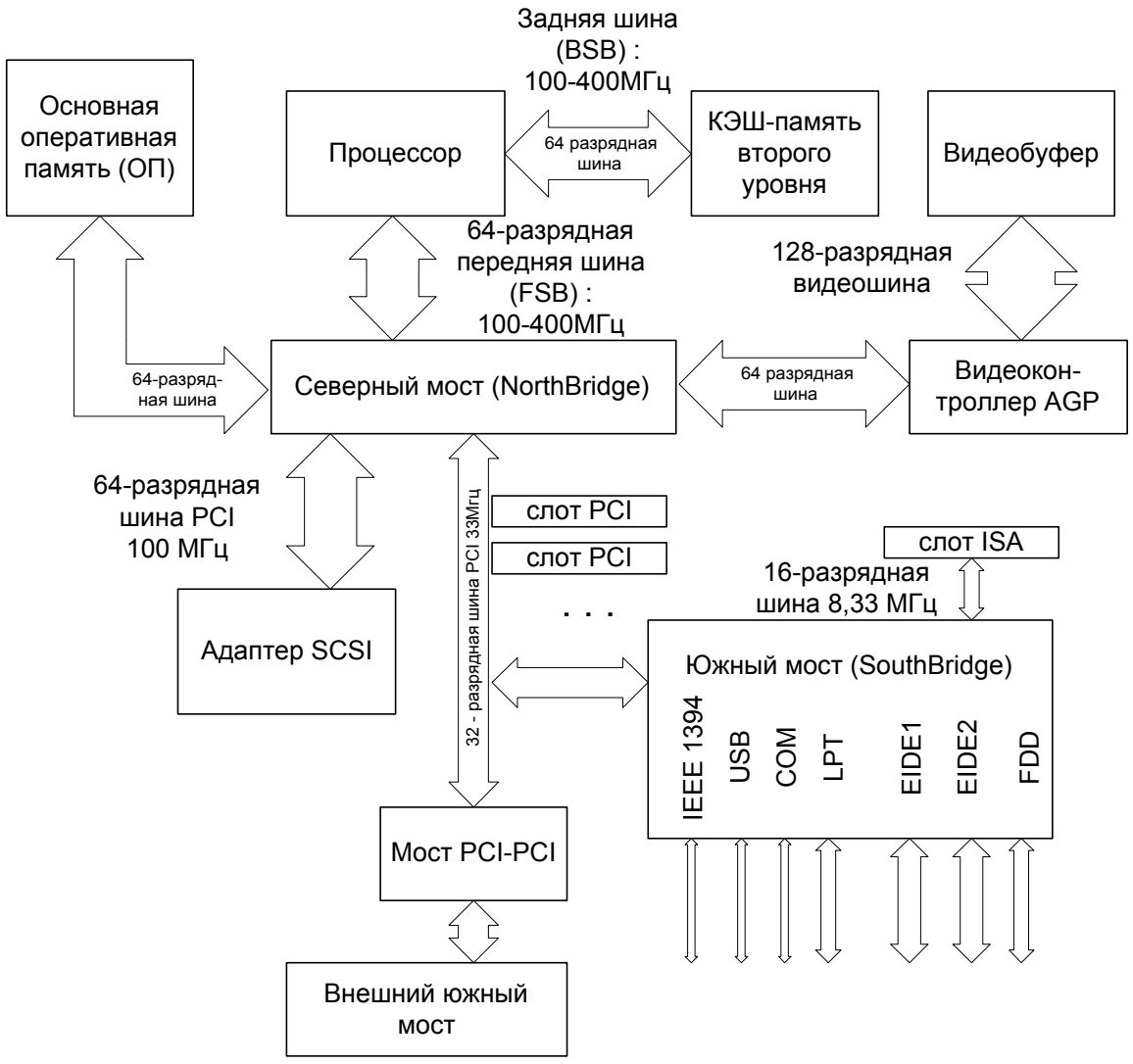


Рис. 4.1

5 ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

5.1. Классификация параллельных ВС

При построении параллельных вычислительных систем (ВС) разработчиками могут ставиться различные цели, такие, например, как :

- повышение производительности;
- улучшение показателя производительность / стоимость;
- повышение надежности функционирования (системы высокой готовности) и т.д.

Параллельные вычислительные машины и системы могут классифицироваться по различным признакам. К наиболее распространенным можно отнести следующие классификации:

1. По взаимодействию потоков команд (инструкций) и потоков данных. Данная классификация предложена американским ученым Флинном (Flinn) в начале 70-х годов и используется до настоящего времени. Он предложил подразделять все ВС на 4 группы :

- ОКОД - Одиночный поток команд / Одиночный поток данных (SISD - Single Instruction / Single Data). Это ВС и ЭВМ обычного последовательного типа (фон-Неймановкой архитектуры). Для данных ЭВМ параллельная обработка реализуется в виде многозадачной обработки (системы с разделением времени и др.). При этом в данный момент времени ЦП или ОУ занято выполнением какой-то одной задачи.

- ОКМД - Одиночный поток команд / Множество потоков данных (SIMD - Single Instruction / Multiple Data). Такая архитектура характерна для векторных и матричных ВС, выполняющих специальные векторные и матричные операции как паралельные операции для разных потоков данных. Под потоками данных подразумеваются последовательности элементов векторов (для векторных ВС) или строки матриц (для матричных ВС). В последние годы SIMD-расширения реализованы в системах команд процессоров общего назначения (MMX, SSE, SSE2 – Intel, 3DNow! – AMD, AltiVec – Motorola и др.)

- МКОД - Множество потоков команд / Одиночный поток данных (MISD - Multiple Instruction / Single Data). Данная архитектура соответствует ВС конвейерного типа, в которых один поток данных проходит разные ступени обработки в разных процессорных элементах (ПЭ).

Архитектуры типа ОКМД и МКМД используются при построении высокопроизводительных систем разного уровня, начиная от простых конвейерных ВС до супер-ЭВМ с векторными и параллельными процессорами.

- МКМД - Множество потоков команд / Множество потоков данных (MIMD - Multiple Instruction / Multiple Data). Такая архитектура характерна для ВС сверхвысокой производительности, в которых множество ПЭ, выполняющих каждый свою вычислительную подзадачу (процесс), обмениваются потоками команд и данных в разных направлениях (транспьютерные системы, системы с массовым параллелизмом и др. – рис. 5.1.)

Помимо четырех выделенных групп, иногда выделяют дополнительные, находящиеся на границе между перечисленными, например, MSIMD или MMISD – соответственно Multi-SIMD, или Multi-MISD – системы с несколькими параллельно работающими SIMD или MISD - блоками.

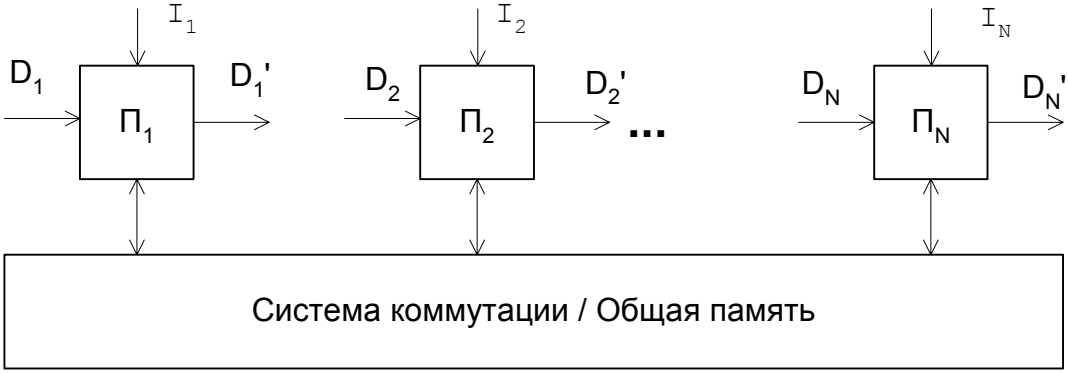


Рис. 5.1

2. По управляющему потоку
- управляемые потоком команд (IF- instruction flow) ;
- управляемые потоком данных (DF- dataflow) .

Системы с управлением потоком данных иногда называют просто потоковыми архитектурами (подробнее они рассмотрены в п. 5.5)

3. По использованию памяти :
- с общей памятью («разделяемая память» - shared memory);
- с локальной памятью для каждого процессора.

Общая разделяемая разными процессорами память может быть физически распределена по вычислительной системе. С другой стороны, под распределенной памятью (distributed) понимают обычно систему, при которой каждый процессор имеет свою локальную память, с локальным адресным пространством. Для общей памяти, доступ к которой осуществляется разными процессорами в системе за одинаковое время, существует термин UMA – Unified Memory Access (память с одинаковым временем доступа). В случае, когда время доступа к разным адресам общей памяти для разных процессоров неодинаково (обычно это характерно для физически распределенной общей памяти), используют термин Non-UMA (память с различным временем доступа) .

4. По способу обмена между процессорами:
- через общую разделяемую память;
- через передачу сообщений.

5. По используемому типу параллелизма :
- естественный или векторный параллелизм (используется в векторных и матричных вычислительных системах);
- параллелизм независимых ветвей («крупнозернистый» или «Coarse Grain») – используется в симметричных многопроцессорных системах Symmetric MultiProcessor (SMP), а также в системах MIMD;

- «мелкозернистый» («Fine Grain») параллелизм – используется в многопроцессорных системах типа MIMD, в системах с массовым параллелизмом и др.;
- параллелизм смежных операций (Instruction Level Parallelism – ILP) - используется в ЭВМ с длинным командным словом – VLIW и др.

6. По способу загрузки данных:
- с последовательной загрузкой (последовательным кодом - по битам) ;
- с параллельной загрузкой (по словам);
- с последовательно-параллельной загрузкой.

7. По системе коммутации :
- полносвязанные МПВС (каждый процессор связан с каждым);
- с выделенным коммутатором ;
- с общей шиной ;
- линейный или матричный массивы (связаны друг с другом соседние процессоры) ;
- гиперкубы (связаны соседние ПЭ, но массивы многомерные);
- параллельные машины с изменяемой конфигурацией связей;
- с программируемыми коммутаторами .

8. По сложности системы коммутации (по кол-ву уровней иерархии) :
- с коммутирующей цепью (сетью) - один уровень коммутации ;
- с кластерной коммутацией (когда группы вычислительных устройств объединены с помощью одной системы коммутации, а внутри каждой группы используется другая).

9. По степени распределеннности ВС :
- локальные вычислительные системы ;
- вычислительные комплексы (в том числе – сильносвязанные – с обменом через общую память и слабосвязанные – с обменом через передачу сообщений).
- как вариант вычислительных комплексов – кластерные архитектуры;
- распределенные вычислительные комплексы, в том числе – сети ЭВМ и распределенные кластерные системы.

10. По организации доступа к общим ресурсам:
- симметричные многопроцессорные системы (все процессоры имеют одинаковый доступ к общим ресурсам);
- асимметричные (master-slave) многопроцессорные системы с разными возможностями доступа для разных процессоров.

5.2 Параллельные вычислительные системы типа SIMD. Векторные ВС.

К ВС типа SIMD относят, прежде всего, *ассоциативные* и *векторные ВС. Ассоциативные* ВС строятся, как можно заключить из названия, на базе ассоциативной памяти. Как уже отмечалось ранее, при рассмотрении организации систем памяти, для ассоциативной памяти характерны аппаратные механизмы поиска информации с заданным адресным признаком по различным

мерам сходства и т.н. вертикальная обработка. В ассоциативных вычислительных системах помимо поиска могут аппаратно реализовываться различные механизмы обработки данных, применяемые к данным с подходящими тегами с использованием все той же вертикальной обработки, то есть - параллельно захватывающие все ячейки памяти, или по крайней мере – все строки массива накопителя ассоциативной памяти.

Тем не менее, чаще всего ассоциативные вычислительные системы строятся для решения задач невычислительного характера – поиска, распознавания и т.д. Примером могут служить различные вычислительные системы для поддержки баз данных – в частности, реляционный ассоциативный процессор (RAP). Существуют и проекты собственно вычислительных ассоциативных систем – например, известная ВМ “STARAN”.

К *векторным* относят системы, включающие в свои системы команд специальные векторные (матричные) операции, такие, как векторное и матричное сложение, умножение вектора на матрицу, умножение матрицы на константу, вычисление скалярного произведения, свертки и т.д. Такие ВС относят к классу SIMD, иногда – MISD. Такая неоднозначность связана с наличием двух основных видов векторных систем :

- В векторно-конвейерных системах в основе ускорения вычислений лежит принцип конвейерной обработки последовательности компонент векторов и матриц.
- В векторно – параллельных системах различные компоненты векторов и матриц при выполнении векторных операций обрабатываются параллельно на параллельно работающих ПЭ.

Векторно-параллельные ВС иногда называют «настоящими SIMD», тем самым подчеркивая, что в векторно-конвейерных ВС имеется несколько иной механизм взаимодействия потоков команд и данных, больше характерный для MISD. В то же время логически или функционально и в векторно-конвейерных системах фактически реализуется принцип SIMD (одна инструкция – например, сложение, выполняется для нескольких потоков данных, но только эти потоки как бы выстраиваются в одну очередь на конвейере).

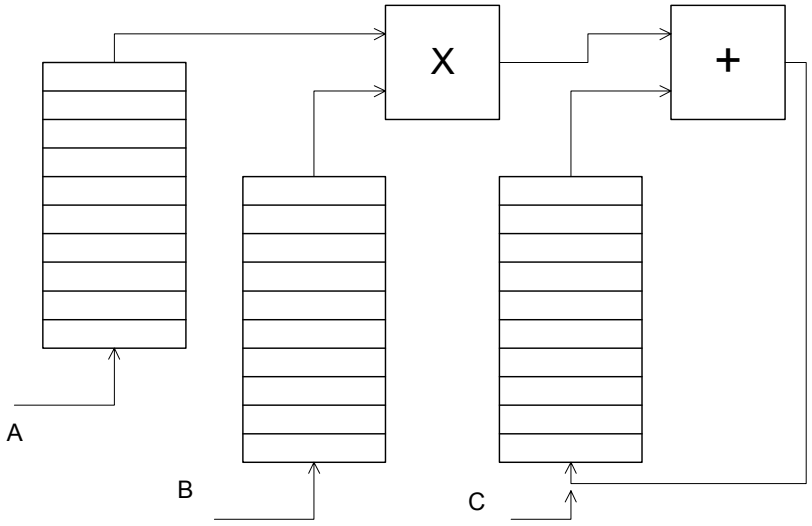


Рис. 5.2

Например, если необходимо находить свертку, то есть вычислять выражение вида :

$$C = \sum_i a_i \cdot b_i \ ,$$

то можно использовать два арифметических конвейера – для умножения и для сложения (сложение тоже

можно конвейеризовать, см. конвейеризацию параллельного сумматора на заключительном этапе формирования произведения в умножителе Брауна), на который подаются последовательно элементы обрабатываемых векторов из векторных регистров (рис. 5.2). Арифметические устройства сами по себе могут быть и неконвейеризованными, тогда либо реализуется конвейер из трех операций: умножение, сложение и запись в регистр (память), либо – три указанные операции просто выполняются как одна макрооперация, которая называется «зацеплением», причем аппаратно ускоряется ее вычисление и подготовка следующего зацепления по сравнению с обычными процессорами общего назначения. Часто зацепление реализуется с использованием трех банков оперативной памяти, а не регистров, поскольку векторные машины должны работать с векторами и матрицами большой размерности, которые не всегда можно разместить в векторных регистрах.

Для ускорения работы с памятью используют различные механизмы адресации, операции с автоинкрементом (автодекрементом) адреса, механизмы ускоренной выборки и записи (многопортовая память, память с расслоением и т.д.), отдельное адресное обрабатывающее устройство (что характерно для так называемой разнесенной архитектуры). Для выполнения скалярных операций в комплексе с векторным обрабатывающим устройством в векторной машине может использоваться скалярное устройство.

Таким образом, для векторно-конвейерных машин характерно :

1. Поддержка специальных векторных, матричных операций в системе команд.
2. Ускорение обработки векторов за счет конвейеризациии выборки и собственно обработки в конвейерных исполнительных устройствах..
3. Наличие векторных регистров.
4. Развитые механизмы адресации и выборки/записи в память.
5. Сочетание векторных и скалярных регистров и обрабатывающих устройств для эффективной реализации алгоритмов, требующих выполнения как векторных, так и скалярных вычислений.

Наряду с конвейеризацией операций в векторных ВС используется и конвейеризация команд, что требует такого построения системы команд векторной машины, чтобы команды легко могли выполняться на конвейере. Поэтому многие векторные процессоры имеют RISC-подобные команды.

Примерами векторно-конвейерных машин могут служить классические супер-ЭВМ серии Cray : Cray-1, Cray-2, Cray – X-MP, Cray – Y- MP и др. Примером векторно-параллельных машин могут служить машины ILIAC-IV, Cyber-205, отечественные супер-ЭВМ серии ПС2000.

Недостатком векторно-параллельных машин является сравнительно низкая эффективность в смысле загрузки процессорных элементов. Высокая производительность достигается только на векторных операциях, в то время как на скалярных операциях и при обработке векторов и матриц меньшей размерности значительная часть устройств может простаивать. В конвейерных ЭВМ при обработке векторов меньшей размерности конвейер, возможно, будет загружен полностью, так как меньшая размерность может компенсироваться

большей интенсивностью следования последовательных по алгоритму векторных подзадач.

Кроме того, программирование векторно-параллельных ЭВМ осуществляется в целом сложнее, чем для векторно-конвейерных. Для загрузки и выгрузки данных требуется больше времени, чем в случае конвейерных систем, когда данные могут поступать последовательно. При этом, преимуществом векторно-параллельных машин является их потенциально более высокая производительность.

В целом векторные машины характеризуются высокой пиковой производительностью при полной загрузке их вычислительных устройств, однако достигается высокая производительность в основном при решении определенного круга задач, эксплуатирующих естественный параллелизм.

5.3 Понятие о систолических структурах и алгоритмах

Под *систолической структурой* можно понимать массив (сеть) вычислительных «клеток» (ПЭ), связанных каналами обмена. Число соседних клеток ограничено. Каждая клетка работает по своей программе (что характерно для систем МКМД). Обмен данными и управляющими сигналами осуществляется только с соседними клетками. Обмен с оперативной памятью происходит только на границе массива клеток (рис. 5.3). Термин «систолическая» (systolic) указывает на синхронность, непрерывность и волнообразность продвижения обрабатываемых данных от одной границы массива к другой (систола – это сердечная мышца, которая работает также синхронно, ритмично и без остановок в ходе всей жизни человека, как систолическая система «прокачивает» через себя обрабатываемые данные). Особенностью работы систолических структур является отсутствие накопления информации в локальных блоках памяти и возвратов потоков данных назад для циклической обработки. Упорядоченность этапов обработки информации указывает на

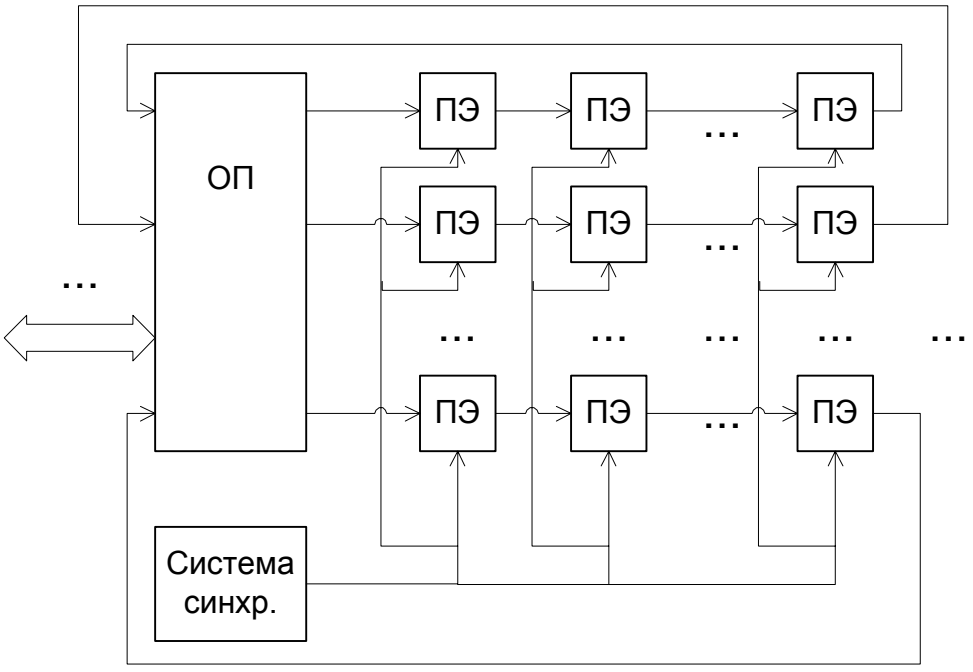


Рис. 5.3

родство с конвейерными архитектурами типа MISD, поэтому многие исследователи относят систолические системы именно к этому классу. В то же время наличие различных потоков команд и данных в системе и разных программ у разных вычислительных клеток указывает на родство с MIMD-архитектурами. В отличие от MIMD характер обмена носит заданный, однонаправленный характер, отсутствие циклов, возвратов, длинных пересылок, а также относительная простота задач, решаемых каждой вычислительной клеткой, позволяет говорить о своеобразии систолических структур.

Областью применения таких структур, прежде всего, являются многопроцессорные специализированные структуры для цифровой обработки сигналов, изображений, для решения матричных задач.

Для эффективной реализации вычислений в систолической структуре необходимы так называемые *систолические алгоритмы*, рассчитанные на аппаратную систолическую реализацию. Они должны удовлетворять определенным требованиям, среди которых :

1) Регулярность, однонаправленность графа вычислений (потокового графа) алгоритма.
2) Ацикличность алгоритма.
3) Возможность разбиения алгоритма на этапы одинаковой сложности и длительности выполнения для построения конвейера.
4) Возможность распараллеливания вычислений.
5) Отсутствие необходимости в больших объемах памяти для сохранения промежуточных результатов и накопления информации.
6) Локальность пересылок информации, отсутствие необходимости в длинных пересылках.
7) Минимальное количество развилок в алгоритме и т.д.
8) Минимальное количество входных и выходных точек алгоритма.
9) Минимальное количество разных типов вычислений и операций, используемых в алгоритме.
10) Возможность разбиения алгоритма на подалгоритмы меньшей размерности, и с другой стороны – наращивания алгоритма для решения задач большей размерности.
11) Гарантированная сходимость вычислений за заданное число шагов (итераций) и др.

Примером систолических алгоритмов являются алгоритмы CORDIC и родственные ему (так называемые ДЛП–алгоритмы или CORDIC–подобные), другие итерационные алгоритмы, алгоритмы обработки матриц, оптимизированные для аппаратной реализации и так далее. Рассмотренные ранее аппаратные умножители также являются примерами систолических структур.

5.4 Масштабируемые параллельные системы МКМД.

Современные параллельные вычислительные системы, включая аппаратные и программные средства, называются *масштабируемыми* (scalable),

если их ресурс может быть изменен в зависимости от требований производительности/стоимости. Это понятие включает в себя следующие составляющие:

- Функциональность и производительность. Масштабируемая система должна повышать (понижать) свою производительность пропорционально повышению (понижению) своих ресурсов. В идеале эта зависимость должна быть линейной.
- Стоимость. Стоимость системы при масштабировании должна меняться пропорционально (но не быстрее, чем линейно).
- Совместимость. Одни и те же компоненты системы, включая аппаратные и программные средства, должны использоваться после масштабирования без больших изменений.

В простейшем случае масштабируемость системы может быть достигнута за счет изменения количества модулей - процессоров или вычислительных машин, входящих в нее. Масштабируемость характеризуется своим размером-максимальным количеством вычислительных модулей, которое может быть включено в нее без нарушения ее работоспособности. Например, Симметричная мультипроцессорная Система (SMP, 1997) имеет предел до 64 процессоров, а система IBM SP2 (1997) – до 512 процессоров.

В настоящее время наибольшее распространение получили пять вариантов МКМД систем:

- Системы с массовым параллелизмом (massively parallel processor-MPP);
- Симметричные мультипроцессорные системы (SMP);
- Кластеры рабочих станций (cluster of workstations-COW);
- Системы с распределенной памятью (distributed shared memory-DSM);
- Систолические структуры.

Системы с массовой параллельной обработкой (MPP) включают тысячи, десятки, иногда - сотни тысяч процессорных элементов. Подобные системы могут иметь различную организацию. Одни системы относят к категории SIMD с мелкозернистым параллелизмом. Они работают под управлением специальных управляющих ЭВМ, раздающих задания процессорам и контролирующих их коммутацию и обмен. Сами процессорные элементы могут быть достаточно простыми и иметь невысокую производительность. В частности, в одной из первых ЭВМ такого типа – Connection Machines – 1 (CM-1) использовались однобитные АЛУ в которых 32-битовое сложение выполнялось за 24мкс! АЛУ объединялись в ПЭ, состоявший из 16 таких ОУ и 4 блоков локальной памяти. Производительность 1 ПЭ - около 2MIPS. Количество процессорных элементов достигало 64 тысяч, разделенных на 4 блока, каждый блок управлялся ЭВМ-секвенсором, отвечавшей за обмен между блоками, а управление всей системой осуществляла фронтальная ЭВМ, транслировавшая высокоуровневый поток команд в поток инструкций для ПЭ. Одна фронтальная ЭВМ могла управлять до 2 млн. процессорных элементов ! Система CM-1 достигала производительности в

100 Gflops (1984 г), а система Connection Machines – 5 с 256 тыс. процессоров – уже до 1 TFlops (1991г).

В других системах с массовой параллельной обработкой процессорные элементы представляют собой достаточно мощные и автономные вычислительные устройства (например, высокопроизводительные процессоры общего назначения), выполняющие собственные программы вычислений. В таких системах может использоваться кластерная организация.

Важную роль в эффективной реализации вычислений в таких системах (как и в параллельных вычислительных системах вообще) играют организация обмена информацией между процессорными элементами, организация памяти и, конечно, программное обеспечение.

Наиболее естественным кажется использовать для обмена общую память (shared memory). При этом адресное пространство памяти доступно всем процессорам, которые помещают результаты своей работы, а также – считывают исходную информацию из общедоступного ресурса (в зависимости от характера и объемов обрабатываемой информации она может храниться в оперативной памяти, либо – во внешней). Как уже отмечалось ранее, общая память может быть физически распределенной (Non-UMA), либо – с одинаковым временем доступа (UMA). Проблемы возникают при синхронизации работы с общими областями памяти (здесь задействуются механизмы семафоров, транзакций, атомарных операций и т.д.), при организации виртуальной памяти, при ускорении доступа к памяти в физически распределенной системе памяти и так далее. Дополнительную сложность представляет синхронизация работы разделяемой оперативной памяти и локальной кэш-памяти процессоров (проблема когерентности кэш-памяти). Синхронизация достигается применением нетривиальных алгоритмов, например, MESI. Для упрощения синхронизации памяти и для ускорения обмена в целом предпочтительнее обмен крупными блоками данных, выполняемый реже, по сравнению с частым обменом мелкими порциями данных. К другим способам ускорения обмена с памятью в многопроцессорных системах можно отнести коммутацию типа «точка-точка», при которой память разбивается на банки, каждый из которых связан с каждым процессором через интеллектуальный коммутатор.

При использовании логически распределенной между процессорами памяти каждый процессор работает со своим адресным пространством, а обмен информации происходит путем передачи сообщений между процессорами. При этом аппаратная организация параллельной системы упрощается, для построения мультипроцессорных систем подойдут даже стандартные телекоммуникационные средства, например, аппаратура и протоколы локальных сетей. С другой стороны, программирование таких систем усложняется, поскольку синхронизация сообщений, управления, потоков данных выполняется в основном программными средствами. В настоящее время используются в основном два базовых программных интерфейса параллельных систем с передачей сообщений : PVM (Parallel Virtual Machine) и MPI (Message Processing Interface).

В целом можно отметить, что степень «зернистости» при разбиении задачи в мультипроцессорных системах может варьироваться из соображений затрат на собственно вычисления и обмен между вычислителями. В некоторых случаях потенциально высокая степень параллелизма, диктующая мелкозернистость, входит в противоречие с большими накладными расходами на организацию такой мелкозернистости, связанными с обменом между процессорами, и тогда меньшее дробление задачи оказывается более предпочтительным.

Среди мультипроцессорных систем большое распространение получили архитектуры, называемые *кластерными*.

Термин «кластерная архитектура» трактуется в настоящее время достаточно широко. Ключевым является понятие кластера как группы каких-то относительно самостоятельных, но тесно взаимодействующих устройств. С этой точки зрения под кластерами понимают, во-первых, вычислительные системы, построенные по иерархическому принципу, использующие кластерную коммутацию, во-вторых – группы автономных вычислительных машин, работающих под управлением общих вычислительных программ (вычислительные кластеры, в том числе – на базе сегментов локальных сетей), в-третьих – серверные системы высокой готовности, включающие несколько (по крайней мере – 2) автономных подсистем и т.д.

Кластеры реализуются как системы с передачей сообщений. Кластеризация позволяет упростить организацию обмена и в целом повысить эффективность системы за счет разумной иерархии и оптимального разделения задач по группам вычислителей.

С точки зрения вычислительной мощности простейшие кластеры как вычислительные комплексы на базе автономных ВМ (в том числе – ПК) могут составлять конкуренцию дорогим вычислительным системам, при этом обладая существенным преимуществом - они намного дешевле. Здесь на первый план выходит эффективное программное обеспечение для параллельной обработки. Чаще всего подобные вычислительные кластеры строятся на базе систем, работающих под управлением ОС UNIX.

Анализ списка 500 самых производительных вычислительных машин, который публикуется каждые 3 месяца в Интернете (адрес: http://www.top500.org), показывает, что верхние строчки списка, как правило, занимают мультипроцессорные MIMD – системы. Например, это компьютеры, строящиеся в рамках программы ASCI (Advanced Strategic Computer Initiative – Стратегическая компьютерная инициатива). Они включают, обычно, десятки тысяч процессоров (общего назначения, либо – специализированных), объединенных в группы, с развитой иерархической системой коммутации. Управляются такие машины специализированным программным обеспечением для поддержки параллельных вычислений.

5.5 Потоковые вычислительные системы

Под потоковыми ВС понимают обычно ВС, управляемые потоком данных (Data Flow). Это довольно интересный класс вычислительных систем, в которых очередной поток вычислений в соответствии с алгоритмом инициируется не очередными инструкциями программы, а готовностью к обработке необходимых данных. С каждой операцией в программе связан набор ее операндов, которые снабжаются флагами готовности к обработке. При установке всех флагов операция отправляется на выполнение. Потенциально такой подход позволяет достичь высокой степени параллелизма, так как потоковая архитектура пытается извлечь максимальный параллелизм из всех заданных вычислений.

Эффективность подобных структур во многом определяется эффективным программированием, задачей которого является формулировка задачи в терминах параллельных и независимых операций. На первый план выходит не эффективное построение процедуры вычислений, а выявление взаимосвязей между потоками данных в задаче. В значительной степени это задача соответствующего оптимизирующего компилятора.

Известно много проектов потоковых вычислительных систем, среди них можно отметить Манчестерскую ВС (Манчестерский университет), Tagged Token, Monsoon (Массачусетский технологический институт), Sigma, EMS, EMC-4 (Тсукуба, Япония), RAPID (проект Sharp – Mitsubishi – университет Осаки, Япония) и др. Как утверждает ряд авторов, многие интересные проекты Data Flow не были по достоинству оценены производителями вычислительной техники, в силу нетрадиционности подхода.

С другой стороны, потоковые вычисления нашли применение в современных суперскалярных процессорах и процессорах с длинным командным словом. Причем если в VLIW – структурах задача выявления параллельных потоков в большей степени решается программным обеспечением, то в суперскалярных процессорах логика управления многопотоковым конвейером как раз и реализует механизм, напоминающий управление потоком данных, но средствами самого процессора. Действительно, последовательный поток инструкций программы в суперскалярном процессоре транслируется в параллельные потоки внутренних инструкций, операнды которых снабжаются признаками готовности, которые, в том числе, зависят и от работы механизма динамической оптимизации команд.

Так что системы, управляемые потоком данных, так или иначе, находят применение в современной вычислительной технике.

Литература :

1. Пятибратов А. Вычислительные системы, сети и телекоммуникации – М., Финансы и статистика, 2002.
2. Каган Б.М. Электронные вычислительные машины и системы.- М.: Энергоатомиздат, 1985.
3. Нортон П., Гудмен Д. Внутренний мир персональных компьютеров, 8-е издание. Избранное от Питера Нортона: Пер. с англ. – К.: Издательство “ДиаСофт”, 1999. – 584 с.
4. Таненбаум Э.С. Архитектура компьютера, 4-е издание – С-Пб.:”Питер-пресс”, 2002. –704с.
5. Столингс У. Структурная организация и архитектура компьютерных систем, 5-е изданеие. – М.: Изд. дом Вильямс, 2002. – 896с.
6. Корнеев В.В., Киселев А.В. Современные микропроцессоры. - М.: “Нолидж”, 2000.- 320с., ил.
7. Корнеев В.В. Параллельные вычислительные системы. – М.: “Нолидж”, 1999.- 320с., ил.
8. Пом А., Агравал О. Быстродействующие системы памяти.- М.:Мир, 1987.
9. Шнитман В. Современные высокопроизводительные компьютеры.: Материалы Центра информационных технологий при МГУ - www.citforum.ru.
10. Кун. С. Матричные процессоры на СБИС. – М.: Мир, 1991.
11. Архитектура IA-32. www.intel.ru, www.developer.intel.ru
12. Архитектура IA-64. www.intel.ru, www.developer.intel.ru
13. Савельев В.А. Прикладная теория цифровых автоматов.-М.: Высшая школа,1988.
14. Духнич Е.И., Лукьянов В.С. Основы синтеза операционных и управляющих автоматов. Учебное пособие; ВолгГТУ, 1991.
15. Егунов В.А. Системы памяти. Учебное пособие; ВолгГТУ, 2000 г.
16. Murdocca M., Heuring V. Principles Of Computer Architecture, 1999., Prentice Hall. (http://www.cs.rutgers.edu/~murdocca/POCA/POCA.html)

ЕВГЕНИЙ ИВАНОВИЧ ДУХНИЧ

АНДРЕЙ ЕВГЕНЬЕВИЧ АНДРЕЕВ

ОРГАНИЗАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ МАШИН И СИСТЕМ

Учебное пособие

Темплан 2003. Поз. № . _____

Редактор _____

Лицензия ЛР № 020251 от 16.04.1996.

Подписано в печать _____ Формат 60x84 1/16.

Бумага газетная. Печать офсетная.

Усл. печ.л. _____. Уч-изд. л. _____ Тираж 200 экз.

Заказ _____.

Волгоградский государственный технический университет.
400131, г. Волгоград, пр. Ленина, 28.
РПК «Политехник» Волгоградского государственного технического университета.
400131, г. Волгоград, ул. Советская, 35.