Проект комиссии Президента
по модернизации и технологическому развитию экономики России
«Создание системы подготовки высококвалифицированных кадров
в области суперкомпьютерных технологий и
специализированного программного обеспечения»

Конспект лекций дисциплины

«Архитектура и программное обеспечение высокопроизводительных вычислительных систем»

Разработчик: к.ф.-м.н. Богословский Н.Н.

Москва

Тема № 1
Параллелизм на уровне архитектуры процессоров

Ранее уже были рассмотрены основные составляющие центрального процессора. В данной теме основное внимание уделено вопросам общей архитектуры процессоров как единого устройства и способам повышения их производительности.

Архитектуры ЦП с полным и сокращенным набором команд.

Современная технология программирования ориентирована на языки высокого уровня (ЯВУ), главная задача которых — облегчить процесс написания программ. Более 90% всего процесса программирования осуществляют на ЯВУ. К сожалению, операции, характерные для ЯВУ, отличаются от операций, реализуемых машинными командами. Эта проблема получила название *семантического разрыва* и ведет она к недостаточно эффективному выполнению программ. Пытаясь преодолеть семантический разрыв, разработчики ВМ расширяют систему команд, дополняя ее командами, реализующими сложные операторы ЯВУ на аппаратурном уровне, вводят дополнительные виды адресации и т. п. Вычислительные машины, где реализованы эти средства, принято называть ВМ с полным набором команд (CISC — Complex Instruction Set Computer). К типу CISC можно отнести практически все ВМ, выпускавшиеся до середины 80-х годов и значительную часть из выпускаемых компьютеров в настоящее время.

Характерные для CISC способы решения проблемы семантического разрыва, вместе с тем ведут к усложнению архитектуры ВМ, главным образом устройства управления, что, в свою очередь, негативно сказывается на производительности в целом. Кроме того, в CISC очень сложно организовать эффективный конвейер команд, который является одним из наиболее перспективных путей повышения производительности ВМ. Все это заставило более внимательно проанализировать программы, получаемые после компиляции с ЯВУ. Был предпринят комплекс исследований, в результате которых обнаружились интересные закономерности:

- Реализация сложных команд, эквивалентных операторам ЯВУ, требует увеличения емкости управляющей памяти в микропрограммном УУ.
- В откомпилированной программе операторы ЯВУ реализуются в виде процедур (подпрограмм), поэтому на операции вызова процедуры и возврата из нее приходится от 15 до 45% вычислительной нагрузки.
- При вызове процедуры вызывающая программа передает этой процедуре некоторое количество аргументов. В 98% случаев число передаваемых аргументов не превышает шести. Примерно такое же положение сложилось и с параметрами, которые процедура возвра-

щает вызывающей программе. Более 80% переменных, используемых программой, являются локальными, то есть создаются при входе в процедуру и уничтожаются при выходе из нее. Количество локальных переменных, создаваемых отдельной процедурой, в 92% случаев не превышает шести.

- Почти половину операций в ходе вычислений составляет операция присваивания, сводящаяся к пересылке данных между регистрами, ячейками памяти или регистрами и памятью.

Детальный анализ результатов исследований привел к серьезному пересмотру традиционных архитектурных решений, следствием чего стало появление архитектуры с сокращенным набором команд (RISC - Reduced Instruction Set Computer) . Термин «RISC» впервые был использован Паттерсоном и Дитцелем в 1980 году.

RISC архитектуры

Главные усилия в архитектуре RISC направлены на построение максимально эффективного конвейера команд, то есть такого, где все команды извлекаются из памяти и поступают в ЦП на обработку в виде равномерного потока, причем ни одна команда не должна находиться в состоянии ожидания, а ЦП должен оставаться загруженным на протяжении всего времени. Кроме того, идеальным будет вариант, когда любой этап цикла команды выполняется в течение одного тактового периода, Последнее условие относительно просто можно реализовать для этапа выборки. Необходимо лишь, чтобы все команды имели стандартную длину, равную ширине шины данных, соединяющей ЦП и память. Унификация времени исполнения для различных команд — значительно более сложная задача, поскольку наряду с регистровыми существуют также команды с обращением к памяти.

Помимо одинаковой длины команд, важно иметь относительно простую подсистему декодирования и управления: сложное устройство управления (УУ) будет вносить дополнительные задержки в формирование сигналов управления. Очевидный путь существенного упрощения УУ — сокращение числа выполняемых команд, форматов команд и данных, а также видов адресации.

Излишне напоминать, что в сокращенном списке команд должны оставаться те, которые используются наиболее часто. Исследования показали, что 80-90% времени выполнения типовых программ приходится на относительно малую часть команд (10-20%). К наиболее часто выполняемым действиям относятся пересылка данных, арифметические и логические операции. Основная причина, препятствующая сведению всех этапов цикла команды к одному тактовому периоду, - потенциальная необходимость доступа к памяти для выборки операндов и/или записи результатов. Сле-

дует максимально сократить число команд, имеющих доступ к памяти. Это соображение добавляет к ранее упомянутым принципам RISC еще два;

- доступ к памяти во время исполнения осуществляется только командами «Чтение» и «Запись»;
- все операции, кроме «Чтение» и «Запись», имеют тип «регистр-регистр».

Для упрощения выполнения большинства команд и приведения их к типу «регистр-регистр требуется снабдить ЦП значительным числом регистров общего назначения. Большое число регистров в регистровом файле ЦП позволяет обеспечить временное хранение промежуточных результатов, используемых как операнды в последующих операциях, и ведет к уменьшению числа обращений к памяти, ускоряя выполнение операций. Минимальное число регистров, равное 32, принято как стандарт де-факто большинством производителей RISC-компьютеров,

Суммируя сказанное, концепцию RISC-компьютера можно свести к следующим положениям:

- выполнение всех (или, по крайней мере, 75% команд) за один цикл;
- стандартная однословная длина всех команд, равная естественной длине слова, ширине шины данных и допускающая унифицированную поточную обработку всех команд;
- малое число команд (не более 128);
- малое количество форматов команд (не более 4);
- малое число способов адресации (не более 4);
- доступ к памяти только посредством команд «Чтение» и -«Запись»;
- все команды, за исключением «Чтения» и «Записи», используют внутрипроцессорные межрегистровые пересылки;
- устройство управления «жесткой» логикой;
- относительно большой (не менее 32) процессорный файл регистров общего назначения (согласно [210] число РОН в современных RISC-микропроцессорах может превышать 500).

Преимущества и недостатки RISC

Сравнивая достоинства и недостатки CISC и RISC, невозможно сделать однозначный вывод о неоспоримом преимуществе одной архитектуры над другой. Для отдельных сфер использования ВМ лучшей оказывается та или иная. Тем не менее, ниже приводится основная аргументация «за» и «против» RISC-архитектуры.

Для технологии RISC характерна сравнительно простая структура устройства управления. Площадь, выделяемая на кристалле микросхемы для реализации УУ, существенно меньше. Так, в RISC I она составляет 6%,а в RISC II - 10%. Как следствие, появляется возможность разместить

на кристалле большое число регистров ЦП (138 в RISC II). Кроме того, остается больше места для других узлов ЦП и для дополнительных устройств: кэш-памяти, блока арифметики с плавающей запятой, части основной памяти, блока управления памятью, портов ввода/вывода.

Унификация набора команд, ориентация на потоковую конвейерную обработку, унификация размера команд и длительности их выполнения, устранение периодов ожидания в конвейере — все эти факторы положительно сказываются на общем быстродействии. Простое устройство управления имеет немного вентилей и, следовательно, короткие линии связи для прохождения сигналов управления. Малое число команд, форматов и режимов приводит к упрощению схемы декодирования, и оно происходит быстрее. Применяемое в RISC УУ с «жесткой» логикой быстрее микропрограммного. Высокой производительности способствует и упрощение передачи параметров между процедурами. Таким образом, применение RISC ведет к сокращению времени выполнения программы или увеличению скорости, за счет сокращения числа циклов на команду.

Простота УУ, сопровождаемая снижением стоимости и повышением надежности, также говорит в пользу RISC. Разработка УУ занимает меньше времени. Простое УУ будет содержать меньше конструктивных ошибок и поэтому более надежно.

Многие современные CISC-машины, такие как VAX 11/780, VAX-8600, имеют много средств для прямой поддержки функций ЯВУ, наиболее частых в этих языках (управление процедурами, операции с массивами, проверка индексов массивов, защита информации, управление памятью и т. д.). RISC также обладает рядом средств для непосредственной поддержки ЯВУ и упрощения разработки компиляторов ЯВУ, благодаря чему эта архитектура в плане поддержки ЯВУ ни в чем не уступает CISC,

Недостатки RISC прямо связаны с некоторыми преимуществами этой архитектуры. Принципиальный недостаток - сокращенное число команд: на выполнение ряда функций приходится тратить несколько команд вместо одной в CISC. Это удлиняет код программы, увеличивает загрузку памяти и трафик команд между памятью и ЦП. Недавние исследования показали, что RISC-программа в среднем на 30% длиннее CISC-программы, реализующей те же функции.

Хотя большое число регистров дает существенные преимущества, само по себе оно усложняет схему декодирования номера регистра, тем самым увеличивается время доступа к регистрам,

УУ с «жесткой» логикой, реализованное в большинстве RISC-систем, менее гибко, более склонно к ошибкам, затрудняет поиск и исправление ошибок, уступает при выполнении сложных команд.

Однословная команда исключает прямую адресацию для полного 32-битового адреса, поэтому ряд производителей допускают небольшую часть команд двойной длины, например в Intel 80960.

Конвейеризация вычислений

Совершенствование элементной базы уже не приводит к кардинальному росту производительности ВМ. Более перспективными в этом плане представляются архитектурные приемы, среди которых один из наиболее значимых — *конвейеризация.*

Общая идея конвейера связана с разбиением некоторого процесса обработки объектов на несколько независимых этапов и организацией одновременного (параллельного) выполнения этих этапов обработки различных объектов, передвигающихся по конвейеру от одного этапа к другому. При движении объектов по конвейеру на разных его участках выполняются разные операции, а при достижении каждым объектом конца конвейера он окажется полностью обработанным. Конвейеры применяются как при обработке команд, так и в арифметических операциях. Для эффективной реализации конвейера должны выполняться следующие условия:

- система выполняет повторяющуюся операцию;
- эта операция может быть разделена на независимые части, степень перекрытия которых невелика;
- трудоёмкость подопераций примерно одинакова.

Количество подопераций называют *глубиной конвейера*. Важным условием нормальной работы конвейера является отсутствие конфликтов, то есть данные, подаваемые в конвейер, должны быть независимыми.

Принцип конвейерной обработки используется в практической жизни достаточно часто. Рассмотрим упрощённый вариант конвейерной обработки объекта на примере изготовления конфеты. Процесс изготовления конфеты можно разбить на следующие этапы: формирование основы конфеты (начинки), глазировка конфеты шоколадом, покрытие конфеты орехами, обёртывание конфеты в индивидуальную упаковку. Предположим, что на выполнение каждого из этапов необходимо потратить 1 минуту. Тогда на изготовление одной конфеты требуется 4 минуты.

Пусть требуется изготовить 100 конфет. Подсчитаем необходимое время изготовления при последовательном (когда все этапы изготовления конфеты выполняются последовательно) и при конвейерном способе обработки. При последовательном способе обработки необходимо умножить время изготовления одной конфеты на количество конфет; как нетрудно посчитать, потребуется 400 минут для изготовления 100 конфет. Теперь рассчитаем время изготовления 100 конфет при использовании конвейерной обработки. На изготовление одной конфеты всё также требуется 4 минуты, но выполнение каждого этапа проходит параллельно и поэтому время, необходимое на изготовление 100 конфет, составляет 103 минуты. Рассмотрим более подробно сам процесс и посчитаем необходимое время. Представим, что мы пронумеровали все конфеты, даже если они ещё не го-

товы. Запускаем наш конвейер и начинаем производить необходимые действия. Сначала помещаем 1-ю конфету на первый этап. После того как пройдёт 1 минута, первый этап выполнится, и конфета под номером один перейдёт на второй этап, при этом одновременно на первый этап поступит конфета номер 2. После 2-й минуты конфета номер 1 перейдёт на 3-й этап, конфета номер 2 – на второй, а на первый этап попадёт конфета номер 3. После 3-й минуты конфеты совершат очередной переход. Первая конфета перейдёт на 4-й этап, вторая конфета – на 3-й, третья конфета – на 2-й этап, а на первый поступит 4-я конфета. После того как пройдёт 4-я минута, и у нас все конфеты, находящиеся на конвейере, перейдут к следующему этапу, первая конфета, которая пройдёт 4-й, последний этап, будет готова. Далее, после окончания очередной минуты у нас будет готова следующая конфета. Так после 5 минут будет готово 2 конфеты, после 6 минут будет готово 3 конфеты и т. д. Как нетрудно посчитать, после того как пройдёт 102 минуты, у нас будет готово 99 конфет, и 100 конфет будет готово через 103 минуты. Как видно из данного примера, при организации конвейерной обработки и выполнении всех действий параллельно можно уменьшить время выполнения общей задачи.

Конвейерные процессоры.

Процессоры современных компьютеров используют особенную технологию - конвейеры, которые позволяют обрабатывать более одной команды одновременно или выполнять конвейерную обработку данных.

Конвейер команд

Идея конвейера команд была предложена в 1956 году академиком С. А Лебедевым. Как известно, цикл команды представляет собой последовательность этапов. Возложив реализацию каждого из них на самостоятельное устройство и последовательно соединив такие устройства, мы получим классическую схему конвейера команд. Выделим в цикле команды шесть этапов:

1. Выборка команды (ВК). Чтение очередной команды' из памяти и занесение ее в регистр команды;
2. Декодирование команды (ДК). Определение кода операции и способов адресации операндов;
3. Вычисление адресов операндов (ВА). Вычисление исполнительных адресов каждого из операндов в соответствии с указанным в команде способом их адресаций;
4. Выборка операндов (ВО). Извлечение операндов из памяти. Эта операция не нужна для операндов, находящихся в регистрах;
5. Исполнение команды (ИК). Исполнение указанной операции;
6. Запись результата (ЗР), Занесение результата в память.

Все этапы команды задействуются только один раз и всегда в одном и том же порядке: одна за другой. Это, в частности означает, что если первая микрокоманда выполнила свою работу и передала результаты второй, то для выполнения текущей команды она больше не понадобиться, и, следовательно, может приступить к выполнению следующей команды. Выделим каждую команду в отдельную часть устройства и расположим их в порядке выполнения. В первый момент времени выполняется первая микрокоманда. Она завершает свою работу и начинает выполняться вторая микрокоманда, в то время как первая готова для выполнения следующей инструкции. Первая инструкция может считаться выполненной, когда завершать работу все пять микрокоманд.

Такая технология обработки команд носит название конвейерной обработки. Каждая часть устройства называется ступенью конвейера, а общее число ступеней – длиной конвейера.

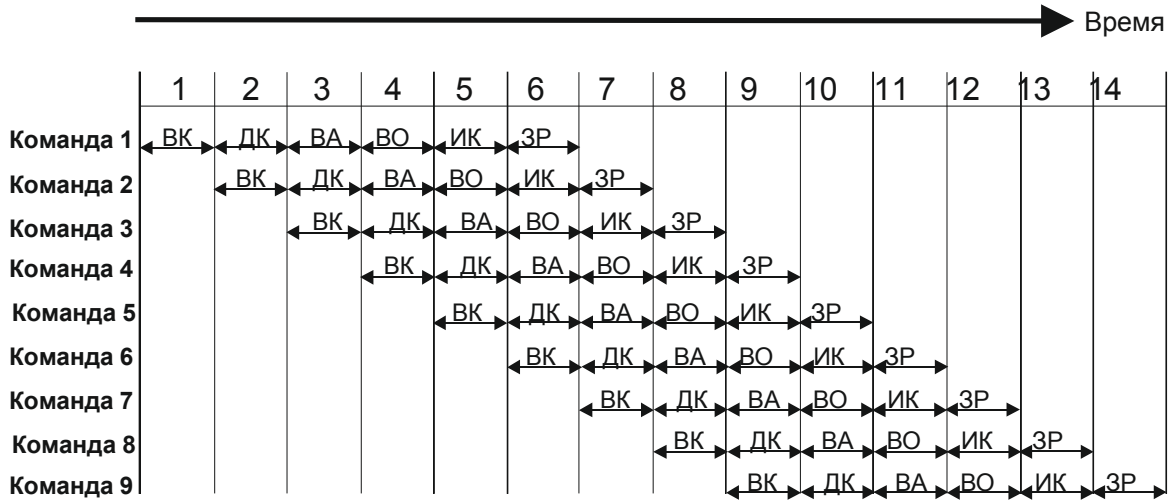

Рис. 1. Логика работы конвейера команд

На рис. 1 показан конвейер с шестью ступенями, соответствующими шести этапам цикла команды. В диаграмме предполагается, что каждая команда обязательно проходит все шесть ступеней, хотя этот случай не совсем типичен. Так, команда загрузки регистра не требует этапа ЗР. Кроме того, здесь принято, что все этапы могут выполняться одновременно. Без конвейеризации выполнение девяти команд заняло бы 9 x 6 = 54 единицы времени. Использование конвейера позволяет сократить время обработки до 14 единиц.

Конвейерная обработка данных.

Во многих вычислительных системах, наряду с конвейером команд, используются и конвейеры данных.

Арифметические действия, производимые процессором с числами, также можно разбить на отдельные этапы и организовать конвейерный принцип работы. Рассмотрим пример машинной команды умножения чисел с плавающей точкой (действительные числа). Все числа в ЭВМ записываются в двоичном формате и хранятся в особом виде.

Выполнение машинного сложения можно разбить на следующие этапы:

- сравнению порядков,
- сдвигу мантиссы меньшего из чисел,
- сложению мантисс
- нормализации результата.

Таким образом, команда сложения может быть разделена на четыре этапа, которые могут быть реализованы в процессоре аппаратно в виде четырёх операционных блоков.

Процессоры первых компьютеров выполняли все эти "микрооперации" для каждой пары аргументов последовательно, одна за одной, до тех пор, пока не доходили до окончательного результата, и лишь после этого переходили к обработке следующей пары слагаемых.

Пусть в операции можно выделить четыре микроопераций, каждая из которых выполняется за одну единицу времени. Если есть одно неделимое последовательное устройство, то 100 пар аргументов оно обработает за 400 единиц. Если каждую микрооперацию выделить в отдельный этап (или иначе говорят - ступень) конвейерного устройства, то на четвертой единице времени на разной стадии обработки такого устройства будут находиться первые четыре пар аргументов, а весь набор из ста пар будет обработан за 4+99=103 единицы времени - ускорение по сравнению с последовательным устройством почти в четыре раза (по числу ступеней конвейера).

В общем случае - n операций будут выполнены на l-ступенчатом конвейере за l+n-1 тактов (единиц времени), а последовательно – за l*n единиц времени.

Казалось бы конвейерную обработку можно с успехом заменить обычным параллелизмом, для чего продублировать основное устройство столько раз, сколько ступеней конвейера предполагается выделить. Действительно, 4 устройств из предыдущего примера обработают 100 пар аргументов за 100 единиц времени, что быстрее времени работы конвейерного устройства! Однако! В чем же дело? Ответ прост, увеличив в 4 раза число устройств, мы значительно увеличиваем как объем аппаратуры, так и ее стоимость. Представьте себе, что на автозаводе решили убрать конвейер, сохранив темпы выпуска автомобилей. Если раньше на конвейере одновременно находилась 1000 автомобилей, то действуя по аналогии с предыдущим примером надо набрать 1000 бригад, каждая из которых в состоянии полностью собрать автомобиль от начала до конца, выполнив сотни

разного рода операций, и сделать это за то же время, что машина прежде находилась на конвейере. Представили себестоимость такого автомобиля?

Рассмотренные примеры организации конвейера предназначены для выполнения одной и той же операции обработки однотипного потока данных, в связи с чем такие конвейеры получили название конвейеров данных.

Сочетание двух конвейеров (конвейер обработки команд и конвейер обработки данных) дает возможность достичь очень высокой производительности на определенных классах задач, особенно если используется несколько различных конвейерных процессоров, способных работать одновременно и независимо друг от друга.

Одной из наиболее высокопроизводительных вычислительных конвейерных систем считается CRAY. В этой системе конвейерный принцип обработки используется в максимальной степени. Имеется и конвейер команд, и конвейер арифметических и логических операций. В системе широко применяется совмещенная обработка информации несколькими устройствами.

Векторные функциональные устройства

Понятие вектора и размещение данных в памяти

В средствах векторной обработки под вектором понимается одномерный массив однотипных данных (обычно в форме с плавающей запятой), регулярным образом размещенных в памяти ЭВМ. Если обработке подвергаются многомерные массивы, их также рассматривают как векторы. Такой подход допустим, если учесть, каким образом многомерные массивы хранятся в памяти ВМ. Пусть имеется массив данных A, представляющий собой прямоугольную матрицу размерности 4x5.

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{21}	a_{22}	a_{23}	a_{24}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}

При размещении матрицы в памяти все ее элементы заносятся в ячейки с последовательными адресами, причем данные могут быть записаны строка за строкой или столбец за столбцом. С учетом такого размещения многомерных массивов в памяти вполне допустимо рассматривать их как векторы и ориентировать соответствующие вычислительные средства на обработку одномерных массивов данных (векторов).

Характеристики вектора:

- Шаг по индексу (stride)
- Длина

Рис. 2. Размещение матрицы в памяти компьютера.

Принцип векторной обработки основан на существовании значительного класса задач использующих операции над векторами. Реализация операций обработки векторов на скалярных процессорах с помощью обычных циклов ограничивает скорость вычислений по следующими причинам:

- перед каждой скалярной операцией необходимо вызывать и декодировать скалярную команду;
- для каждой команды необходимо вычислять адреса элементов данных;
- данные должны вызываться из памяти, а результаты запоминаться в памяти. В больших ЭВМ память выполняется, как правило, в виде набора модулей, доступ к которым может осуществляться одновременно. В условиях, когда каждая команда вырабатывает свой собственный запрос к памяти, такой раздробленный доступ может стать причиной возникновения конфликтов обращения к памяти, препятствующих эффективному использованию ее потенциальной пропускной способности;

- необходимо осуществлять упорядочение выполнения операций в функциональных устройствах. В целях увеличения производительности эти устройства строятся по конвейерному принципу. Эффективному использованию конвейерных устройств препятствует последовательная “природа” оператора цикла;
- Реализация команд построения циклов (счетчик и переход) сопровождается накладными расходами. Кроме того, наличие в цикле команды перехода препятствует эффективному использованию принципа опережающего просмотра;

Влияние перечисленных отрицательных факторов уменьшается при введении векторных команд, с помощью которых задается одна и та же операция над элементами одного или нескольких векторов, и организации, системы, которая обеспечивает эффективное исполнение таких команд. Этот подход реализуется в системах двух типов: матричных и векторно-конвейерных.

- Матричная система состоит из множества процессорных элементов (ПЭ), организованных таким образом, что они исполняют векторные, команды, задаваемые общим для всех устройством управления, причем каждый ПЭ работает с отдельным элементом вектора. ПЭ соединены через коммутационное устройство с многомодульной памятью. Исполнение векторной команды включает чтение из памяти элементов векторов, распределение их по процессорам, выполнение заданной операции и засылку результатов обратно в память.
- В векторно-конвейерной системе, напротив, имеется один (или небольшое число) конвейерный процессор, выполняющий векторные команды путем засылки элементов векторов в конвейер с интервалом, равным длительности прохождения одной, стадии обработки. При этом скорость вычислений зависит только от длительности стадии и не зависит от задержек в процессоре в целом.

Оба подхода в принципе позволяют достичь значительного ускорения по сравнению со скалярными машинами. Более того, ускорение в системах матричного типа может быть больше, чем в конвейерных, поскольку увеличить число процессорных элементов проще, чем число ступеней в конвейерном устройстве.

Понятие векторного процессора

Векторный процессор — это процессор, в котором операндами некоторых команд могут выступать упорядоченные массивы данных — векторы.

Рассмотрим возможные подходы к архитектуре средств векторной обработки. Наиболее распространенные из них сводятся к трем группам:

- конвейерное АЛУ;
- массив АЛУ;
- массив процессорных элементов.

Последний вариант - один из случаев многопроцессорной системы, известной как матричная ВС. Понятие векторного процессора имеет отношение к двум первым группам, причем, как правило, к первой.

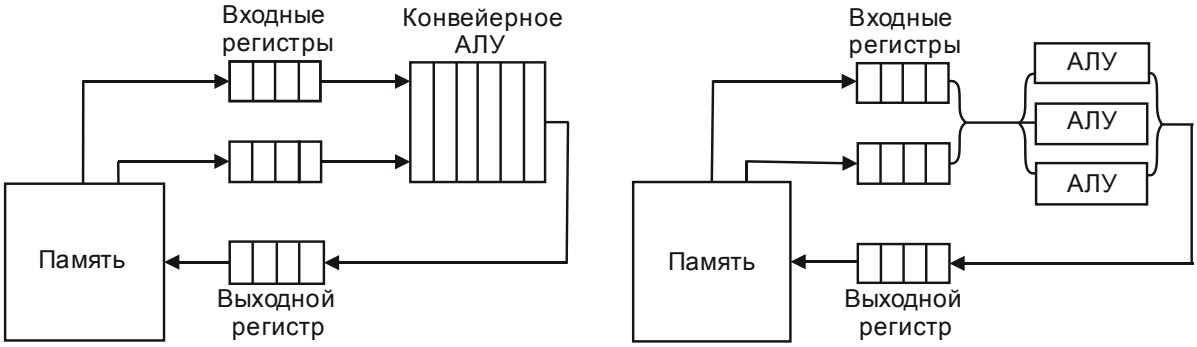

Рис. 3. Схематическое представление устройства векторных процессоров.

В варианте с конвейерным АЛУ (слева) обработка элементов векторов производится конвейерным АЛУ для чисел с плавающей запятой (ПЗ). Операции с числами в форме с ПЗ достаточно сложны, но поддаются разбиению на отдельные шаги. Так, сложение двух чисел может быть сведено к четырем этапам:

1. сравнению порядков,
2. сдвигу мантиссы меньшего из чисел,
3. сложению мантисс
4. нормализации результата.

Каждый этап может быть реализован с помощью отдельной ступени конвейерного АЛУ. Очередной элемент вектора подается на вход конвейера, как только освобождается первая ступень. Ясно, что такой вариант вполне годится для обработки векторов.

Одновременные операции над элементами векторов можно проводить и с помощью нескольких параллельно используемых АЛУ, каждое из которых отвечает за одну пару элементов.

Если параллельно используются конвейерные АЛУ, то возможен еще один уровень конвейеризации. Вычислительные системы, где реализована эта идея, называют векторно-конвейерными. Коммерческие векторно-конвейерные ВС, в состав которых для обеспечения универсальности включен также скалярный процессор, известны как суперЭВМ.

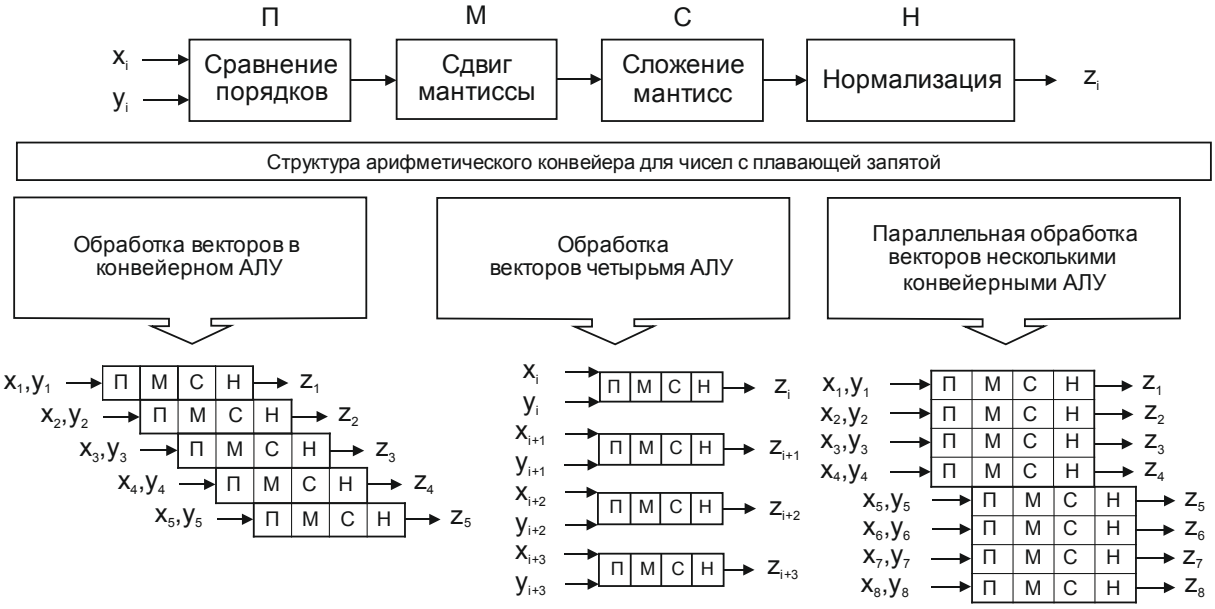

Рис. 4. Способы обработки арифметических операций.

Область применения векторно-конвейерных ВС – задачи моделирования реальных процессов и объектов, для которых характерна обработка больших регулярных массивов чисел в форме с плавающей запятой. Такие массивы представляются матрицами и векторами, а алгоритмы их обработки описываются в терминах матричных операций.

Для обработки массивов требуются вычислительные средства, позволяющие с помощью единой команды производить действие сразу над всеми элементами массивов – средства **векторной обработки**.

Достоинства векторного процессора

1. Вместо многократной выборки одних и тех же команд достаточно произвести выборку только одной векторной команды, что позволяет сократить издержки за счет устройства управления и уменьшить требования к пропускной способности памяти.
2. Векторная команда обеспечивает процессор упорядоченными данными. Когда инициируется векторная команда, ВС знает, что ей нужно извлечь **n** пар операндов, расположенных в памяти регулярным образом. Таким образом, процессор может указать памяти на необходимость начать извлечение таких пар. Если используется память с чередованием адресов, эти пары могут быть получены со скоростью одной пары за цикл процессора и направлены для обработки в конвейеризированный функциональный блок. При отсутствии чередования адресов или других средств извлечения операндов с высокой скоростью преимущества обработки векторов существенно снижаются.

Суперскалярные процессоры

Поскольку возможности по совершенствованию элементной базы уже практически исчерпаны, дальнейшее повышение производительности ВМ лежит в плоскости архитектурных решений. Как уже отмечалось, один из наиболее эффективных подходов в этом плане - введение в вычислительный процесс различных уровней параллелизма. Ранее рассмотренный конвейер команд - типичный пример такого подхода. Тем же целям служат и арифметические конвейеры, где конвейеризации подвергается процесс выполнения арифметических операций. Дополнительный уровень параллелизма реализуется в векторных и матричных процессорах, но только при обработке многокомпонентных операндов типа векторов и массивов. Здесь высокое быстродействие достигается за счет одновременной обработки всех компонентов вектора или массива, однако подобные операнды характерны лишь для достаточно узкого круга решаемых задач. Основной объем вычислительной нагрузки обычно приходится на скалярные вычисления, то есть на обработку одиночных операндов, таких, например, как целые числа. Для подобных вычислений дополнительный параллелизм реализуется значительно сложнее, но тем не менее возможен и примером могут служить суперскалярные процессоры.

Суперскалярным (этот термин впервые был использован в 1987 году) называется центральный процессор (ЦП), который одновременно выполняет более чем одну скалярную команду. Это достигается за счет включения в состав ЦП нескольких самостоятельных функциональных (исполнительных) блоков, каждый из которых отвечает за свой класс операций и может присутствовать в процессоре в нескольких экземплярах.

Структура типичного суперскалярного процессора показана на рис. 5, Процессор включает в себя шесть блоков: выборки команд, декодирования команд, диспетчеризации команд, распределения команд по функциональным блокам, блок исполнения и блок обновления состояния.

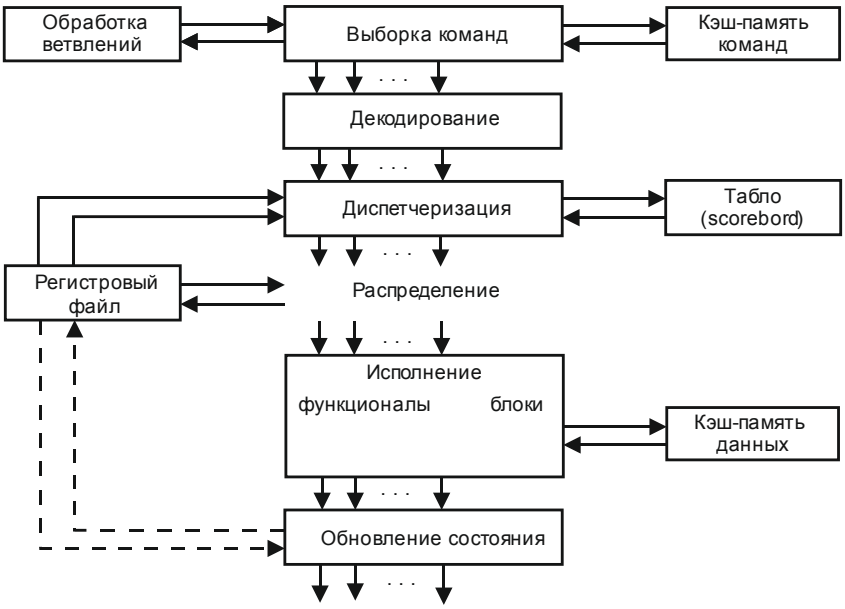

Рис. 5. Архитектура суперскалярного процессора.

Блок выборки команд извлекает команды из основной памяти через кэш-память команд. Этот блок хранит несколько значений счетчика команд и обрабатывает команды условного перехода.

Блок декодирования расшифровывает код операции, содержащийся в извлеченных из кэш-памяти командах, В некоторых суперскалярных процессорах, например в микропроцессорах фирмы Intel, блоки выборки и декодирования совмещены.

Блоки диспетчеризации и распределения взаимодействуют между собой и в совокупности играют в суперскалярном процессоре роль контроллера трафика. Оба блока хранят очереди декодированных команд. Очередь блока распределения часто рассредоточивается по несколько самостоятельным буферам — накопителям команд или схемам резервирования (reservation station), — предназначенным для хранения команд, которые уже декодированы, но еще не выполнены. Каждый накопитель команд связан со своим функциональным блоком (ФБ), поэтому число накопителей обычно равно числу ФБ, но если в процессоре используется несколько однотипных ФБ, то им придается общий накопитель. По отношению к блоку диспетчеризации накопители команд выступают в роли виртуальных функциональных устройств. Оба вида очередей показаны на рис. 6. В некоторых суперскалярных процессорах они объединены в единую очередь.

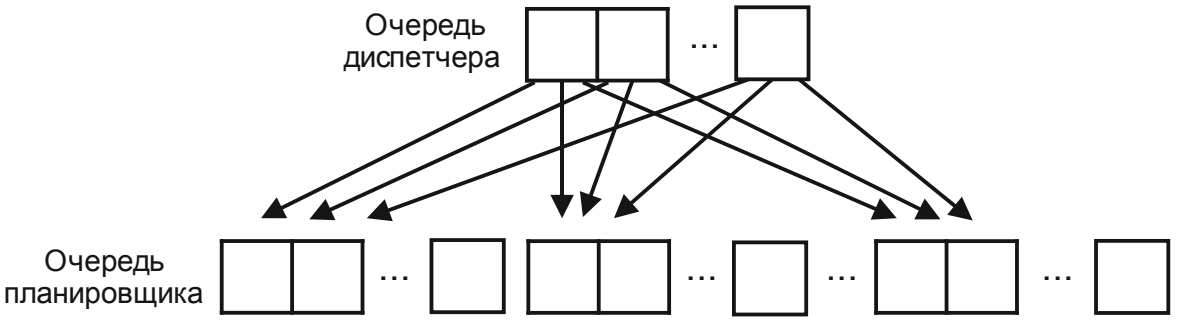

Рис. 6. Очереди диспетчеризации и распределения

В дополнение к очереди, блок диспетчеризации хранит также список свободных функциональных блоков, называемый табло (Scoreboard). Табло использует при диспетчеризации, блок диспетчеризации извлекает команды из своей очереди, считывает из памяти или регистров операнды этих команд, после чего, в зависимости от состояния табло, помещает команды и значения операндов в очередь распределения. Эта операция называется выдачей команд. Блок распределения в каждом цикле проверяет каждую команду в своих очередях на наличие всех необходимых для ее выполнения операндов и при положительном ответе начинает выполнение таких команд в соответствующем функциональном блоке.

Блок исполнения состоит из набора функциональных блоков. Примерами ФБ могут служить целочисленные операционные блоки, блоки умножения и сложения с плавающей запятой, блок доступа к памяти.

Когда исполнение команды завершается, ее результат записывается и анализируется блоком обновления состояния, который обеспечивает учет полученного результата теми командами в очередях распределения, где этот результат выступает в качестве одного из операндов.

Как было отмечено ранее, суперскалярность предполагает параллельную работу максимального числа исполнительных блоков, что возможно лишь при одновременном выполнении нескольких скалярных команд. Последнее условие хорошо сочетается с конвейерной обработкой, при этом желательно, чтобы в суперскалярном процессоре было несколько конвейеров, например два или три.

Подобный подход реализован в микропроцессоре Intel Pentium, где имеются два конвейера, каждый со своим АЛУ (рис. 7). Отметим, что здесь, в отличие от стандартного конвейера, в каждом, цикле необходимо производить выборку более чем одной команды. Соответственно, память ВМ должна допускать одновременное считывание нескольких команд и операндов, что чаще всего обеспечивается за счет ее модульного построения.

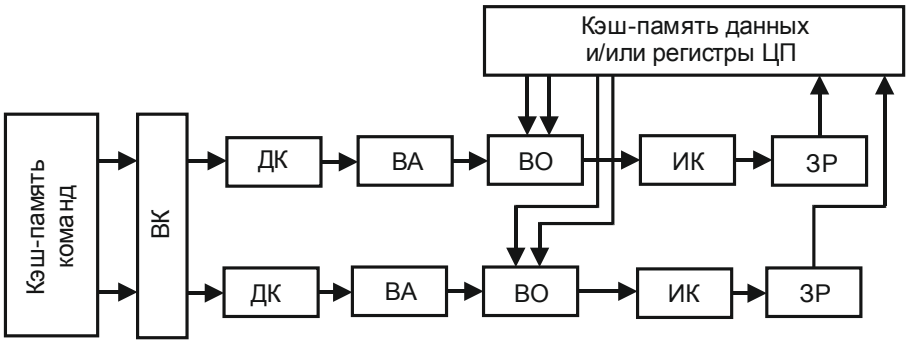

Рис. 7. Суперскалярный процессор с двумя конвейерами

Более интегрированный подход к построению суперскалярного конвейера показан на рис, 8. Здесь блок выборки (ВК) извлекает из памяти более одной командах и передает их через ступени декодирования команды и вычисления адресов операндов в блок выборки операндов (ВО). Когда операнды становятся доступными, команды распределяются по соответствующим исполнительным блокам. Обратим внимание, что операции «Чтение», «Запись» и «Переход» реализуются самостоятельными исполнительными блоками. Подобная форма суперскалярного процессора используется в микропроцессорах Pentium II и Pentium III фирмы Intel, а форма с тремя конвейерами - в микропроцессоре Athlon фирмы AMD.

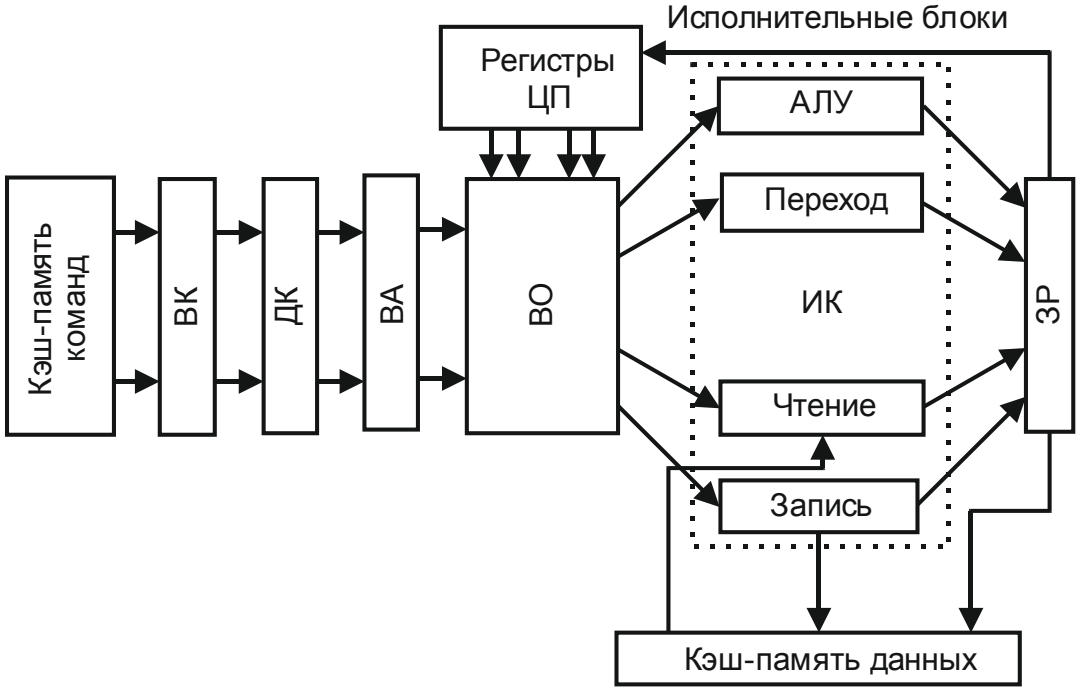

Рис. 8. Суперскалярный конвейер со специализированными исполнительными блоками

По разным оценкам, применение суперскалярного подхода приводит к повышению производительности ВМ в пределах от 8 до 18 раз.

Для сравнения эффективности суперскалярного и суперконвейерного режимов на рис. 9 показан процесс выполнения восьми последователь-

ных скалярных команд. Верхняя диаграмма иллюстрирует суперскалярный конвейер, обеспечивающий в каждом тактовом периоде одновременную обработку двух команд. Отметим, что возможны суперскалярные конвейеры, где одновременно обрабатывается большее количество команд.

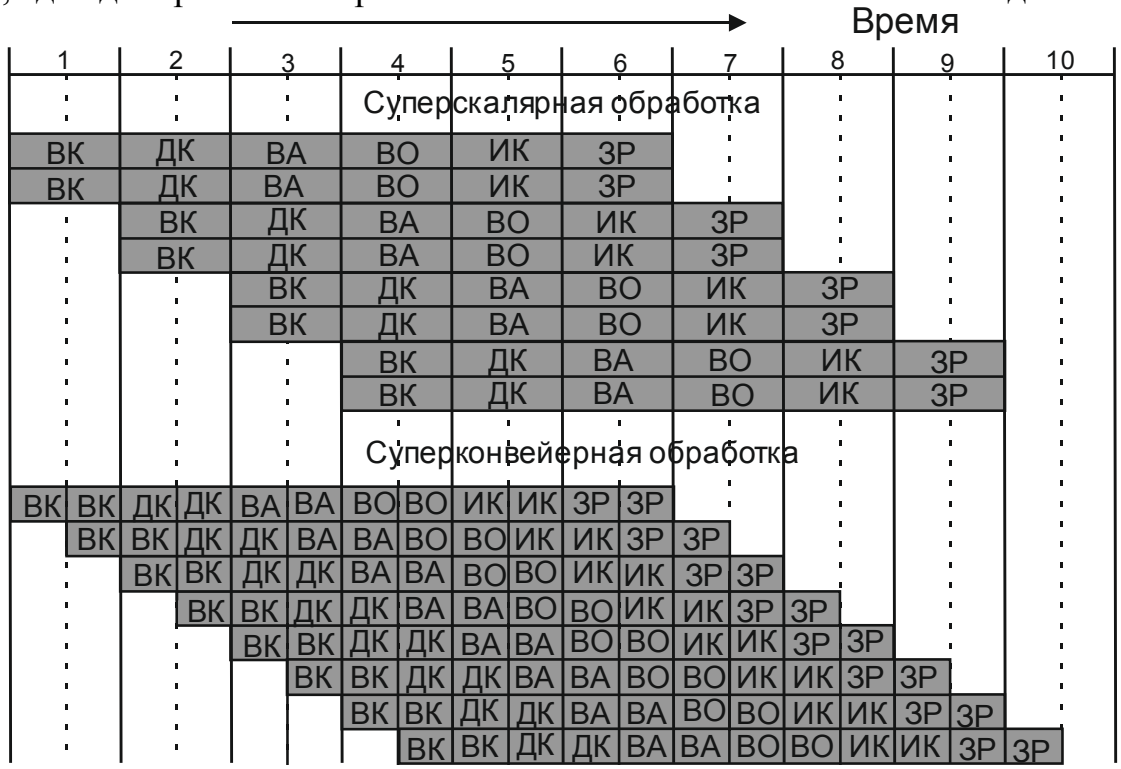

Рис. 9. Сравнение суперскалярного и сулерконвейерного подходов

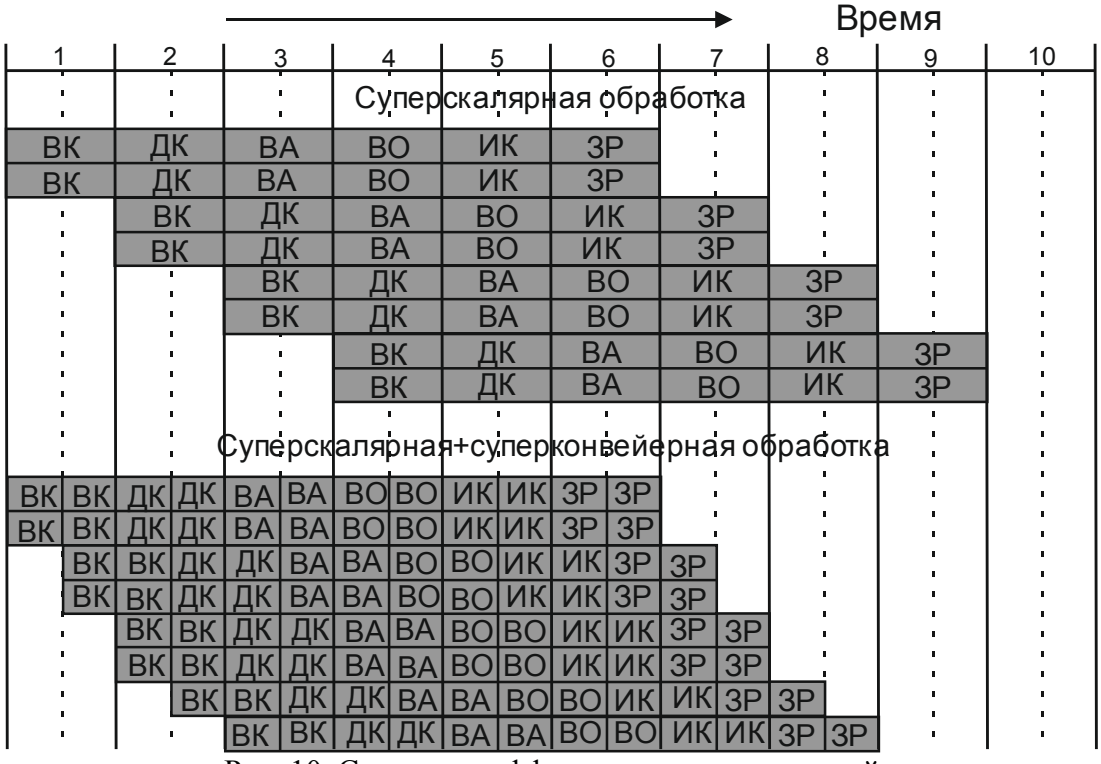

Рис. 10. Сравнение эффективности стандартной суперскалярной и совмещенной схем суперскалярных вычислений

В процессорах некоторых ВМ реализованы как суперскалярность, так и суперконвейеризация (рис. 10). Такое совмещение имеет место в микропроцессорах Athlon и Duron фирмы AMD, причем охватывает оно не только конвейер команд, но и блок обработки чисел в форме с плавающей запятой.

Вычислительные системы с командными словами сверхбольшой длины (VLIW)

Архитектура с командными словами сверхбольшой длины или со сверхдлинными командами (VLIW, Very Long Instruction Word) известна с начала 80-х из ряда университетских проектов, но только сейчас, с развитием технологии производства микросхем она нашла свое достойное воплощение. VLIW - это набор команд, организованных наподобие горизонтальной микрокоманды в микропрограммном устройстве управления.

Идея VLIW базируется на том, что задача эффективного планирования параллельного выполнения нескольких команд возлагается на «разумный» компилятор. Такой компилятор вначале исследует исходную программу с целью обнаружить все команды, которые могут быть выполнены одновременно, причем так, чтобы это не приводило к возникновению конфликтов. В процессе анализа компилятор может даже частично имитировать выполнение рассматриваемой программы. На следующем этапе компилятор пытается объединить такие команды в пакеты, каждый из которых рассматривается так одна сверхдлинная команда. Объединение нескольких простых команд в одну сверхдлинную производится по следующим правилам:

- количество простых команд, объединяемых в одну команду сверхбольшой длины, равно числу имеющихся в процессоре функциональных (исполнительных) блоков (ФБ);
- в сверхдлинную команду входят только такие простые команды, которые исполняются разными ФБ, то есть обеспечивается одновременное исполнение всех составляющих сверхдлинной команды.

Длина сверхдлинной команды обычно составляет от 256 до 1024 бит. Такая метакоманда содержит несколько полей (по числу образующих ее простых команд) , каждое из которых описывает операцию для конкретного функционального блока. Сказанное иллюстрирует рис, 11, где показан возможный формат сверхдлинной команды и взаимосвязь между ее полями и ФБ, реализующими отдельные операции.

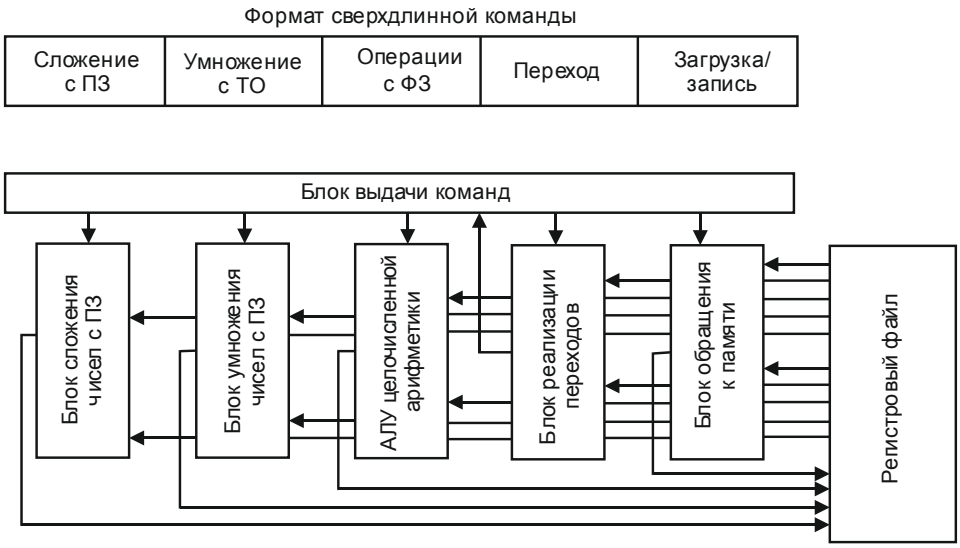

Рис. 11. Формат сверхдлинной команды и взаимосвязь полей команды с составляющими блока исполнения

Как видно из рисунка, каждое поле сверхдлинной команды отображается на свой функциональный блок, что позволяет получить максимальную отдачу от аппаратуры блока исполнения команд.

VLIW-архитектуру можно рассматривать как статическую суперскалярную архитектуру. Имеется в виду, что распараллеливание кода производится на этапе компиляции, а не динамически во время исполнения. То, что в выполняемой сверхдлинной команде исключена возможность конфликтов, позволяет предельно упростить аппаратуру VLIW-процессора и, как следствие, добиться более высокого быстродействия.

В качестве простых команд, образующих сверхдлинную,- обычно используются команды RISC-типа, поэтому архитектуру VLIW иногда называют постRISC-архитектурой. Максимальное число полей в сверхдлинной команде равно числу вычислительных устройств и обычно колеблется в диапазоне от 3 до 20. Все вычислительные устройства имеют доступ к данным, хранящимся в едином многопортовом регистровом файле. Отсутствие сложных аппаратных механизмов, характерных для суперскалярных процессоров (предсказание переходов, внеочередное исполнение и т. д.), дает значительный выигрыш в быстродействии и возможность более эффективно использовать площадь кристалла. Подавляющее большинство цифровых сигнальных процессоров и мультимедийных процессоров с производительностью более 1 млрд операций/с базируется на VLIW-архитектуре. Серьезная проблема VLIW - усложнение регистрового файла и связей этого файла с вычислительными устройствами.