Основы мультиклеточной архитектуры

Функционирование любой вычислительной системы определяется программой или совокупностью программ, каждая из которых задает последовательность операций, выполняемых соответствующим процессором (процессорным устройством), входящим в состав данной системы. Программа состоит из организованного множества информационно связанных команд, размещенных в определенном порядке в памяти системы и содержащих закодированные указания процессору (машинный код) о выполнении требуемых операций.

Являясь частью содержимого памяти, команда представлена в ней в виде командного слова – непрерывной последовательности одноразрядных элементов памяти, занимающей часть ячейки (минимального адресуемого участка памяти), либо целую ячейку, либо несколько ячеек памяти. Состояние элементов памяти командного слова определяет выполняемую операцию и используемые при этом операнды.

Каждая команда форматирована, т.е. имеет определенную структуру, в соответствии с которой ее командное слово логически разделено на несколько частей (полей). В этих полях кодируются: выполняемая операция; операнды (номера регистров, адреса ячеек памяти, номера каналов ввода/вывода и т.п.) используемые данной операцией, а также служебная информация.

Определение содержательной составляющей команд, а именно, выполняемой операции и операндов, с использованием которых она должна быть выполнена, а также кодирование программы, включающее организацию множества команд в единое целое – программу, размещение их в памяти и кодирование в соответствии с существующим форматированием – осуществляются в процессе программирования. Как правило, для этой цели используются разнообразные средства программирования, реализованные на вычислительных системах общего назначения (компьютерах), которые обеспечивают либо запись алгоритма решаемой задачи на языке высокого уровня, например, C, C++, Фортран и т.п., с последующей

компиляцией на машинно-ориентированный язык ассемблера, либо непосредственное программирование на языке ассемблера.

Как известно, язык ассемблера – это система обозначений, используемая для представления в удобочитаемой форме программ, записанных в машинном коде. Программа на языке ассемблера транслируется ассемблером, также реализованном на компьютере, непосредственно в машинный код, который затем записывается в память вычислительной системы каким либо известным способом с помощью устройства.

В зависимости от выбранных средств программирования, кодирование программы выполняется в процессе компиляции (в случае использования языков высокого уровня), либо непосредственно программистом при написании программы на языке ассемблера.

Результат выполнения процессором любой команды выражается в изменении состояния ряда устройств процессора и/или системы (регистров, шин, ячеек памяти, каналов ввода/вывода и т.д.). Использование этого нового состояния последующими исполняемыми командами, образует информационную связь между командой–источником результата и командами–приемниками этого результата. Эта информационная связь может быть как непосредственной (прямой), так и опосредованной (косвенной).

В случае прямой информационной связи команды именуются и при этом имена должны идентифицировать собственно команду, а не ее местоположение или другие особенности реализации. Доступ к результатам может осуществляться по именам как команд–источников, так команд–приемников. В первом случае используется широковещательная рассылка с последующим поименным отбором, при которой в поле операнда команды–приемника задают имя команды–источника. Во втором – выполняется поименная рассылка, при которой в команде–источнике задают имя команды–приемника результата.

При косвенной информационной связи между командами результат доступен только после его отчуждения. А именно, после записи его командой–источником в общедоступные регистры и/или память, имя которых при использовании задают в поле операнда команды–приемника, или в какие-то другие устройства процессора

и/или вычислительной системы, доступ к которым имеют только конкретные команды–приемники. Например, при адресной рассылке результат записывается по указанным в команде–источнике адресам памяти, в которых находятся команды–приемники и в указанное поле этих команд.

Используемые виды информационной связи между командами и характер доступа к результатам являются ключевыми факторами, определяющими архитектурную модель процессора. Именно они определяют не только способы кодирования программ, но также способы и устройства их исполнения.

Известен целый ряд процессорных устройств, использующих косвенную информационную связь, но отличающихся характером доступа к результатам и, соответственно, архитектурной моделью (см. Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем: Учебник для ВУЗов (изд: 2) – СПб,: Питер, 2011). К данным устройствам относятся фон-неймановские, потоковые и редукционные процессоры.

На базе фон-неймановской модели построены все коммерчески успешные процессоры. В ее основе лежит общедоступность результатов и, как следствие, решение процессором определенной задачи достигается пошаговым целенаправленным изменением состояния вычислительной системы. Этот процесс изменения состояний является результатом исполнения последовательности команд строго в заданном порядке, который определяют в процессе программирования.

Функционально фон-неймановский процессор включает устройство управления, исполнительные устройства, устройство ввода/вывода, Устройство управления осуществляет выборку команды из памяти, ее декодирование и передачу соответствующему исполнительному устройству или устройству ввода/вывода, которые исполняют полученную команду. При этом возможны два способа выборки: принудительный и естественный.

Принудительный способ выборки не предусматривает каких-либо ограничений на размещение команд в памяти, но требует, чтобы в каждом командном слове было выделено поле, в которое должен быть записан адрес следующей исполняемой команды.

При наиболее распространенном естественном способе выборки, команды размещаются в памяти последовательно, одна за другой, в том порядке, в котором

предполагается их исполнение. Поле с адресом последующей исполняемой команды используется только в командах передачи управления, когда необходимо перейти после завершения исполнения одной выполняемой последовательности команд к другой.

Для осуществления естественного способа выборки устройство управления содержит специальный регистр, так называемый «счетчик команд». После записи программы в память в счетчик команд записывается адрес первой исполняемой команды программы. Устройство управления считывает из памяти команду, адрес которой находится в счетчике команд, и помещает ее в регистр команд. Команда декодируется, т.е. определяется выполняемая операция, и подготавливаются операнды, после чего она передается на исполнение. Если выбранная команда не является командой перехода, то значение счетчика команд увеличивается на длину выбранной команды, выраженную в единицах адресации, и процесс повторяется. Если была выбрана команда перехода, то вычисляется адрес перехода и его значение записывается в счетчик команд, после чего выборка и исполнение команд начинаются с данного адреса.

Как правило, все варианты фон-неймановской модели направлены на решение одной и той же задачи, а именно, на повышение производительности процессора, но решают они ее разными способами.

Например, в конвейерных процессорах совмещают выполнение команд во времени при использовании одного исполнительного устройства. В RISC процессорах – сокращают количество функций реализуемых командой и, следовательно, время выполнения одной команды. В CISC процессорах увеличивают объем функций выполняемых одной командой и, соответственно, время, но при этом уменьшают общее количество выполняемых команд.

За счет совмещения выполнения команд в пространстве, т.е. на нескольких исполнительных устройствах, обеспечивают повышение производительности суперскалярные процессоры и процессоры со сверхдлинным командным словом (VLIW процессоры). При этом в суперскалярных процессорах задача распределения потока команд по нескольким исполнительным устройствам решается

преимущественно аппаратным способом, а в VLIW процессорах – программным, во время компиляции.

В конкретных реализациях эти и другие способы могут сочетаться. В частности, RISC и CISC процессоры обычно имеют конвейерную организацию, но отличаются друг от друга длиной конвейера. Конвейер также широко используется для сокращения времени выполнения потока команд поступающих к исполнительному устройству в суперскалярных процессорах. Наиболее совершенные способы организации конвейера не только реализуют совмещение выполнения потока команд во времени, но и обеспечивают переупорядочивание команд в конвейере для устранения конфликтов по данным между ступенями конвейера. В этом случае команда в конвейере выполняется не по очередности, а по готовности.

Известны две схемы, осуществляющие такое переупорядочивание команд. Это схема централизованного окна команд и распределенная схема Томасуло.

Следует отметить, что ни использование методов распараллеливания в суперскалярных или в VLIW процессорах, ни методов переупорядочивания команд в конвейере, которые меняют очередность исполнения команд – не отменяют ключевого принципа фон-неймановской модели – упорядоченного изменения состояния процессора. Вне зависимости от очередности исполнения команд, формируемые ими состояния процессора меняются в той последовательности, в которой эти команды были выбраны из памяти, а не исполнены.

Несмотря на разнообразие реализаций, все процессорные устройства, использующие фон-неймановскую модель архитектуры, имеют один общий недостаток.

Это необходимость решения сложной задачи распараллеливания для организации параллельной работы как внутри процессора, при наличии нескольких однотипных исполнительных устройств, так и при использовании многоядерных процессоров или многопроцессорной вычислительной системы.

В потоковом процессоре используется адресная рассылка результатов, при которой полученный результат рассылается по адресам всех команд–приемников. Типичная команда потокового процессора, например, двухместная арифметическая

операция, содержит, как правило, код операции, поля операндов, поле адреса команды, которой передается результат выполненной операции, и поле номера операнда, в котором он будет размещен. Команда выбирается и выполняется «по готовности операндов», т.е. тогда, когда ею будут получены все результаты выполнения других команд, необходимые для ее выполнения. Таким образом, команды в программе могут быть размещены произвольным образом. Принцип исполнения команд потокового процессора «по готовности операндов» позволяет реализовать параллелизм «естественным» образом, т.е. без необходимости решения задачи распараллеливания, но требует для этого использования специальных языков программирования (языков с однократным присваиванием), а также, по сравнению с фон-неймановской моделью, приводит к резкому увеличению затрат на пересылку операндов и выполнению лишних вычислений.

Многочисленные попытки модифицирования архитектуры потоковых процессоров приводили к решению локальных проблем, не изменяя сути данного метода. Например, ввод тегов для обеспечения реентерабельности подпрограмм или для параллельного выполнения различных итераций цикла является виртуальным расширением адреса («адрес» & «тег»), но не сменой парадигмы.

Учитывая большой объем существующего в настоящее время программного обеспечения, написанного на традиционных императивных языках, необходимость использования языков с однократным присваиванием для программирования потоковых процессоров делает их применение экономически нецелесообразным.

Редукционная модель процессора также использует адресную связь между командами, но она задается указанием в поле операнда команды–приемника адреса команды–источника, которую необходимо выполнить для получения требуемого результата. В этом случае команды выбираются из памяти и исполняются «по запросу команды» Такая модель принципиально не имеет лишних вычислений и также позволяет отказаться от упорядоченности команд, которые могут быть размещены в памяти произвольным образом.

Как и потоковая, редукционная модель обеспечивает «естественную» реализацию параллелизма, но тоже требует применения для программирования специальных языков, а именно языков функционального программирования. Это

требование, как и аналогичное требование к потоковой модели, не позволяет использовать существующее программное обеспечение, что делает использование редукционных процессоров экономически невыгодным. Следует также отметить, что выборка команд «по запросу» ограничивает возможности совмещения по времени процессов выборки и исполнения команд и, как следствие, их параллельную реализацию.

Разноплановость преимуществ, присущих разным моделям, привела к созданию процессорных моделей, сочетающих различные способы реализации параллелизма и управления вычислительным процессом. Наиболее известные из них – это макропотоковые, и их развитие – гиперпотоковые процессоры.

В макропотоковых моделях исполнение «по готовности» относится не к одной команде, а к последовательности из нескольких команд, называемых нитью (thread). При этом последовательность команд в пределах нити статична и определена при компиляции, при этом характер ее выполнения фон-неймановский. Как правило, для этого используется конвейерная организация. Порядок обработки нитей зависит от их готовности и меняется динамически в процессе вычислений.

Известны две формы макропотоковой обработки: без блокирования и с блокированием. В модели без блокирования выполнение нити начинается только после получения всех необходимых данных. При этом выполнение нити не прерывается, она выполняется до конца без остановки. В варианте с блокированием выполнение нити может быть начато до получения всех операндов, при этом, когда требуется отсутствующий операнд, выполнение нити приостанавливается (блокируется). Процессор запоминает всю необходимую информацию о состоянии такой заблокированной нити и начинает выполнение другой готовой нити. После получения отсутствующего операнда и завершения выполнения текущей нити, выполнение блокированной нити возобновляется. Модель с блокированием обеспечивает более полную загрузку исполнительных устройств за счет использования механизма блокировки и, как следствие, возможности формирования длинных нитей.

Гиперпотоковые модели эмулируют работу нескольких логических процессоров на одном вычислительном ядре. При этом в процессе компиляции

формируются такие последовательности команд, которые процессор мог бы обрабатывать параллельно, занимая исполнительные устройства, не занятые одним из потоков, подходящими командами из другого независимого потока. Это принципиальное отличие гиперпотоковых моделей от макропотоковых. При этом характер выполнения потока логическим процессором – фон-неймановский, а управление исполнением потоков осуществляется с помощью операционной системы.

Сочетанием применяемых методов отличается и известный проект процессора TRIPS (см. "The Distributed Microarchitecture of the TRIPS Prototype Processor," K. Sankaralingam, R. Nagarajan, P. Gratz, R. Desikan, D. Gulati, H. Hanson, C. Kim, H. Liu, N. Ranganathan, S. Sethumadhavan, S. Sharif, P. Shivakumar, W. Yoder, R. McDonald, S.W. Keckler, and D.C. Burger, 39th International Symposium on Microarchitecture (MICRO), December, 2006), архитектуру которого авторы относят к EDGE-архитектуре – явного исполнения графа потока данных. Процессор TRIPS имеет два функциональных блока, каждый из которых является фактически VLIW процессором, содержащим 16 (4x4) 64-разрядных арифметико-логических устройств (АЛУ) с плавающей точкой и управляемым потоком данных. От традиционного потокового процессора они отличаются тем, что команды загружаются во все АЛУ одновременно, но процесс их выполнения такой же как в потоковом процессоре – асинхронный, по готовности операндов.

Разные методы используют и в известной синергической вычислительной системе – синпьютере, который как и процессор TRIPS имеет асинхронную организацию, но в отличие от него состоит из множества идентичных, функционально полных вычислительных устройств, объединенных коммутационной средой, обеспечивающей обмен результатами между ними (см. патент РФ № 2179333 на изобретение «Синергетическая вычислительная система», дата подачи 13.06.2000 г., опубликовано 10.02.2002 г., Streltsov N., Sparso J., Bokov S., Kleberg S. The Synputer – A Novel MIMD Processor Targeting High Performance Low Power DSP Applications // International Signal Processing Conference, Dallas, 1-3 April, 2003). Для программирования синпьютера алгоритм рассматривается как совокупность формул, представленных в ярусно-параллельной форме, а именно, в

виде ациклического ориентированного графа, вершины которого взаимно-однозначно соответствуют используемым в формуле величинам и операциям, а дуги соответствуют и отражают направление информационных связей между ними. При этом вершины распределены по ярусам (подмножествам) так, что вершины одного яруса непосредственно не связаны друг с другом, а ярусы пронумерованы таким образом, что вершины-источники всегда размещены на вышележащем ярусе (имеющем номер меньше).

Местоположение любого узла ярусно-параллельной формы задается парой чисел (i,k), где: i – номер яруса (строки); k – номер узла в ярусе (столбца).

В общем случае, для выполнения операций, расположенных на i-ом ярусе, могут использоваться результаты, полученные на ярусах от 1-го до (i-1)-го. В синергической вычислительной системе эта возможность ограничена следующим образом:

для выполнения операций на i-ом ярусе используются только результаты (i-1)-го яруса;

если для выполнения операции на i-ом ярусе нужен результат с яруса j, где $1 \leq j < (i-1)$, то на ярусе j+1 этот результат запоминается в регистровой или оперативной памяти, а на ярусе i-1 читается.

Это ограничение позволяет исключить из указания источника результата номер яруса. Источник однозначно задается номером узла в предшествующем ярусе и, таким образом, для кодирования двухместных операций достаточно двух полей операндов, содержащих номера узлов источников, результаты которых используются в качестве аргументов при выполнении данной операции. Например, в таблице 1 приведена программа синпьютера для вычисления формулы g=e*(a+b)+(a-c)/f.

Таблица 1

Номер яруса	**Номер узла**			
	1	2	3	4
1	чт a	чт b	чт c	
2	+1,2	чт e	-1,3	чт f

3	* 1,2	/ 3,4		
4	+ 1,2			
5	зп 1,g			

В синпьютере, последовательность описаний операций одного узла является упорядоченной последовательностью команд, выполняемых вычислительным устройством с соответствующим номером. При этом операции одного яруса в принципе могут размещаться произвольно – в любом узле. На них не накладывается требование упорядоченности. Возможность неупорядоченного размещения команд в одном ярусе упрощает решение задачи распараллеливания, но не позволяет реализовать параллелизм «естественным» путем.

Каждое вычислительное устройство состоит из памяти программ, памяти данных, блока регистров общего назначения и процессора, включающего устройство управления и несколько попарно связанных буферных и исполнительных устройств.

Устройство управления реализует классическую фон-неймановскую схему выбора команд, т.е. их последовательность определяется счетчиком команд. Буферные устройства обеспечивают хранение выбранных команд до получения всех операндов и передачу готовых команд на исполнение. Исполнительные устройства выполняют готовые команды и передают их результат в коммутационную среду.

В процессе выборки команд устройства управления всех вычислительных устройств согласованно выбирают команды очередного яруса и присваивают им значения тегов, равные последовательному номеру выбранного яруса. Декодируют выбранные команды и, при необходимости, формируют заявку коммутационной среде на необходимые им результаты вычислительных устройств, полученные при выполнении команд предыдущего яруса, указывая коммутационной среде номер тега предыдущего яруса и номера вычислительных устройств, после чего записывают выбранную команду, в буферное устройство. Выбор буферного устройства определяется выполняемой операцией. Затем формируют новое значение счетчика команд, и процесс выборки повторяется.

В буферном устройстве команда будет находиться до тех пор, пока она не будет сформирована, а именно, пока коммутационная среда не получит и не передаст в буфер все запрошенные при выборке данной команды результаты.

Процесс формирования команд инициализируется поступлением в коммутатор очередного результата. При этом коммутационная среда по тегу результата проверяет наличие заявок на данный результат от данного вычислительного устройства. Если заявки есть, то результат передается в буферные устройства соответствующих вычислительных устройств, которые записывают его в поля операндов тех команд, которые его запрашивали.

Процесс исполнения команды и рассылки результатов инициализируется, если в буферном устройстве есть хотя бы одна команда, у которой получены все операнды (исполнение «по готовности»), и которая не является последней выбранной командой (кроме команды остановки). При этом, если готовых к исполнению команд несколько, то на исполнение передается команда, которая из них была выбрана первой. Исполнительное устройство выполняет команду и выдает в коммутатор её результат вместе с тегом команды.

Завершается или приостанавливается этот процесс тогда, когда в буферных устройствах нет готовых к исполнению команд.

Использование принципа исполнения команд «по готовности» в синпьютере и в процессоре TRIPS – не позволяет отнести их к потоковым процессорам. Характер размещения команд и их выборки на исполнение – фон-неймановский. Принцип «по готовности» ограничен только этапом исполнения команд. Соответственно, указанный недостаток фон-неймановской архитектуры присущ обоим процессорам.

Следует также отметить, что ряд общих недостатков имеют все процессорные устройства, использующие косвенную информационную связь с любым принципом доступа к результатам и, соответственно, с любой архитектурной моделью.

Это, во-первых, полная машинная зависимость программ. При изменении конфигурации процессора или вычислительной системы, например, при изменении количества используемых исполнительных устройств, как правило, требуется перекомпиляция программ, а иногда и полное перепрограммирование.

Во-вторых, непрерывное увеличение семантического разрыва между постоянно развиваемыми языками программирования высокого уровня и машинным кодом. В результате повышается сложность компиляторов и, следовательно, затраты на их разработку.

Указанные недостатки, как и недостаток фон-неймановской архитектуры, связанный с необходимостью решения задачи распараллеливания, возникли из-за использования косвенной информационной связи между командами и, следовательно, они неустранимы в процессорных устройствах, использующих эту связь.

Принципиально иной вид информационной связи между командами – это прямая связь. Примером кодирования программы с использованием прямых информационных связей между командами является язык триад, используемый в качестве промежуточного представления исходной программы в процессе компиляции (см. Ахо А., Сети Р., Ульман Д. Компиляторы: принципы, технологии и инструменты – М,: Диалектика, 2003).

Промежуточное представление в виде триад образуется записями с тремя полями:

<op><arg1><arg2>,

где: op – поле кода операции; arg1 – поле первого операнда; arg2 – поле второго операнда.

В качестве операндов используются это либо указатели на таблицу символов (для имен, определенных программистом, или констант), либо указатели на триады–источники используемых результатов.

Для следующей последовательности операторов, написанной на языке высокого уровня и образующей линейный участок:

```
L1:   a:=b+c;
      i:=a*f–g;
      a:=i /h;
      goto L2;
```

промежуточное представление на языке триад будет выглядеть так, как показано в таблице 2.

Таблица 2.

Номер триады	op	arg1	arg2
0	+	b	c
1	*	(0)	f
2	–	(1)	g
3	/	(2)	h
4	:=	i	(2)
5	:=	a	(3)
6	goto	L2	

Имена – это указатели на таблицу символов, а числа в скобках – это указатели (ссылки) на триады, причем допустима только ссылка на триады данного линейного участка. Информационная связь между различными линейными участками – косвенная.

Данная последовательность триад может рассматриваться как фрагмент машинного кода некоторого виртуального процессора, написанного на языке ассемблера. Техническая реализация этого процессора возможна, но малоэффективна. Основным недостатком данного способа кодирования является сквозная нумерация команд в программе. Она требует либо значительных аппаратных затрат на реализацию ссылок, либо ограничений на длину исходной программы. Она также не позволяет одновременно обрабатывать несколько итераций цикла, так как имеет статический характер, и одна и та же операция в различных итерациях имеет одно и то же значение ссылки.

Указанные недостатки устранены в контекстно-зависимой программе, исполняемой мультиклеточным процессором (см. Стрельцов Н.В. Архитектура и реализация мультиклеточных процессоров // Труды V Международной научной конференции «Параллельные вычисления и задачи управления» - Москва, 26-28 октября 2010 г., стр.1087-1103).

Контекстно-зависимая программа представляет собой неупорядоченную совокупность параграфов, каждый из которых является линейным участком.

Информационная связь между командами внутри параграфа – прямая, с широковещательной рассылкой и последующим поименным отбором результатов, а между параграфами связь косвенная, через память. Команды параграфа образуют частично упорядоченную и информационно замкнутую последовательность, а именно, команды–источники предшествуют своим командам-приемникам, а команды–приемники имеют ссылки на команды-источники только своего параграфа.

Ссылки кодируются числом, значение которого равно разнице между номерами команды-приемника и команды-источника, полученными при последовательной нумерации команд параграфа. Так, например, фрагмент программы, приведенный в таблице 2, будет выглядеть следующим образом (см. таблицу 3).

Таблица 3.

Номер команды	**op**	**arg1**	**arg2**
0	rd		b
1	rd		c
2	rd		f
3	rd		g
4	rd		h
5	+	(–5)	(–4)
6	*	(–1)	(–4)
7	–	(–1)	(–4)
8	/	(–1)	(–4)
9	:=	(–2)	i
10	:=	(–2)	a
11	goto	L2	

Последнюю команду параграфа отмечают управляющим признаком «конец параграфа». Максимальное значение ссылки (окно видимости результата исполненной команды) определяется размером ее поля в команде и равно 2**p–1, где p – размер поля ссылки в битах.

Контекстно-зависимая программа выполняется мультиклеточным процессором, состоящим из N идентичных клеток с номерами от 0 до n–1, объединенных коммутационной средой, блока памяти данных (DM), блока регистров общего назначения (GPR) и периферийных устройств. Каждая клетка содержит блок памяти программ (PM), устройство выборки команд (PMS), устройство управления (CU), буферные устройства (BUF), исполнительные устройства (EU), устройство внутренней рассылки (ICU), а также каждой клетке соответствует коммутационное устройство (SU), совокупность которых образует коммутационную среду (SB).

Процесс исполнения контекстно-зависимой программы мультиклеточным процессором аналогичен процессу исполнения программы в синпьютере. Основные отличия заключаются в размещении контекстно-зависимой программы в памяти программ, определении по ссылке индивидуальных номеров для каждого запрошенного результата и использовании общих блока памяти данных и блока регистров общего назначения.

Для исполнения контекстно-зависимой программы, команды каждого параграфа последовательно, начиная с одного и того же адреса, являющегося адресом данного параграфа, размещают в PM клеток. Адрес размещения первого исполняемого параграфа равен адресу, начиная с которого PMS клеток выбирают команды. В результате, в PM i–ой клетки, начиная с адреса параграфа, последовательно размещаются его команды с номерами i, N+i, 2*N+i,...,.

PMS каждой клетки обеспечивает, начиная с указанного ему адреса, последовательную выборку команд и размещение их на регистре команд для последующего декодирования. Команды клетками выбираются синхронно.

Необходимыми условиями декодирования очередной команды являются завершение декодирования предыдущей команды всеми клетками, считывание и размещение на регистре команд всеми клетками очередной команды и готовность буферных устройств всех клеток к размещению микрокоманд (наличие свободных строк для записи микрокоманд). Если эти условия выполняются, команду декодируют и размещают в буферных устройствах.

При декодировании каждой очередной выбранной группе из N команд присваивают очередное значение тега (t), а команде из группы и, соответственно, ее результату – номер. Значение очередного тега формируется клетками синхронно и изменяется циклически от 0 до T^{max} –1, где T^{max}=2**p. Минимальное значение T^{max}, при котором принципиально невозможна взаимоблокировка команд, определяется размером окна видимости результатов и для N=2**k равно W/N, где W=2**q – размер окна видимости, q – размер поля ссылки в команде в битах.

Занятость тега фиксируют в регистре занятых тегов устройства управления. Если j–тое значение тега присвоено группе команд (тег занят), то в j–тый бит данного регистра записывают единицу. Значение тега освобождают, когда результат команды с этим тегом поступает в коммутатор. Тогда в соответствующий бит записывают ноль, и это значение может быть присвоено другой группе команд.

Номер команды равен t&i, где i – номер клетки, выбравшей данную команду. По значению номера команды и значениям ее ссылок формируют номера запрашиваемых результатов, которые сообщают коммутационному устройству. Номер запрашиваемого результата определяется следующим образом:

$$N_r = ((t\&i) - P_{op}) \bmod (T^{max})$$

где: N_r - номер запрашиваемого результата; P_{op} – значение ссылки. Следует отметить, что младшие k бит N_r содержат номер клетки, от которой должен прийти запрашиваемый результат, а старшие p – k бит содержат значение тега команды, которая формирует данный результат.

Выборка и декодирование команд продолжается до тех пор, пока не будет выбрана команда, отмеченная управляющим признаком «конец параграфа».

Адрес нового параграфа может поступить как в любой момент выборки команд текущего параграфа, так и после завершения выборки команд. Он поступает всем клеткам одновременно. Если к моменту выборки последней команды параграфа адрес следующего параграфа не вычислен, то выборка приостанавливается до получения адреса. Если адрес получен, то выборка продолжается с этого адреса. Выборка также приостанавливается, если нет места в каком-либо буферном устройстве, если занято очередное значение тега, а также,

если по окончанию параграфа не закончилось выполнение команд, формирующих новые значения регистров GPR и не выполнены все записи в память данных.

Буферные устройства формируют команды и передают их соответствующему исполнительному устройству. и состоят из буфера хранения операционной части команды и буфера хранения операндов.

Операционная часть, кроме кода команды, включает в себя всю необходимую служебную информацию для рассылки и приема результатов, а именно, номер команды и признаки готовности первого(второго) операнда для выполнения команды.

Буфер хранения операндов имеет ассоциативную адресацию. Ассоциативным адресом является номер запрашиваемого результата.

В качестве операндов при выполнении операций могут использоваться:

запрошенные результаты команд–источников, поступающие из коммутатора;

значения, вычисленные при декодировании командного слова, а также непосредственно присутствующие в командном слове или взятые из регистров общего назначения.

В первом случае признак готовности данного операнда при записи команды устанавливают в состояние «не готов», а во втором – в состояние «готов». После получения запрошенного результата от коммутационного устройства, признак, в первом случае, также устанавливают в состояние «готов».

Команда, получившая все операнды, проходит приоритетный отбор среди других готовых команд в буферном устройстве, после чего выдается на исполнение при условии незанятости исполнительного устройства.

При записи команды в буфере хранения операционной части устанавливают признак занятости соответствующей строки, который снимают когда команду выдают на исполнение, и, если есть хотя бы одна свободная строка, формируют признак готовности буферного устройства к приему следующей команды.

Доступ к коммутационному устройству осуществляется устройством внутренней рассылки, который обеспечивает выбор наиболее приоритетного результата и выдачу его в коммутатор. Причем, до тех пор пока разница между

номером результата и номером декодируемой в этот момент команды не превысит размер окна видимости, результат задерживают в этом устройстве.

Каждый полученный результат вместе с его тегом посылают во все коммутационные устройства, которые отбирают по номерам, запрошенные данной клеткой результаты, и передают их в ее буферные устройства в качестве операндов.

Каждое коммутационное устройство состоит из N однотипных устройств выбора, порядковый номер которых указывает на клетку, результаты которой отбираются данным устройством выбора и двух блоков передачи операндов. Теги запрошенных результатов хранятся в позиционном виде в памяти запросов устройства выбора. Память запросов представляет собой два регистра размером T^{max} бит, соответственно, для первого и второго операндов. Так при запросе результата с тегом j для использования в качестве первого операнда, в j–том бите первого регистра записывают единицу («запрос установлен»), а после получения результата с этим тегом в j–тый бит записывают ноль («запрос снят»).

В общем случае, каждое коммутационное устройство может одновременно получить до N результатов и все они могут быть отобраны. Для парирования неравномерности их поступления внутри каждого блока передачи операндов используется буферная память.

В этом случае, при поступлении одновременно нескольких отобранных результатов, один или два результата сразу передают в буферные устройства, а остальные записывают в буферную память блока передачи операндов и передают позже. Время задержки результата в буферной памяти определяется интенсивностью и шириной потока, а также принятой дисциплиной обслуживания результатов, находящихся в буферной памяти.

Для сокращения объёма буферной памяти используют блокировку выдачи результатов в коммутационное устройство тем клеткам, в блоках передачи операндов которых переполнена буферная память. Обеспечивается это сигналами занятости, которые формируют все коммутационные устройства.