

# Мультиклеточные процессоры (концепция)



Екатеринбург 2011

**Рынок информационных технологий, общий объем которого составляет триллионы долларов, прямо или косвенно зависит от архитектуры процессора.**

**С момента создания первого компьютера на этом рынке абсолютно доминирует фон-неймановская архитектура. Но, по мнению экспертного сообщества, эпоха этой архитектуры завершается. Динамика развития рынка требует качественно новой (пост-неймановской) архитектуры, способной на десятилетия определить дальнейшее направление создания микропроцессоров.**

**Выход такой архитектуры на рынок создает реальную возможность занятия лидирующих позиций.**

**Исследования процессорных архитектур, проведенные в ООО «УралАрхЛаб», показывают, что такое пост-неймановское направление всего одно(!) – это создание процессоров с контекстно-зависимой программой. Только они могут решить как существующие проблемы, так и перспективные задачи компьютерной индустрии.**

**Разработка, при поддержке Фонда «Инновационные технологии», первого мультিকлеточного процессора с контекстно-зависимой программой МСр0402100100 подтвердила реализуемость и правильность выбранного направления.**

**Разноплановость и качественный состав преимуществ предлагаемой архитектуры, позволяют позиционировать ее как принципиально новое и высокоэффективное пост-неймановское направление развития микропроцессорной техники.**

# СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ .....</b>	<b>4</b>
<b>1 КОНЦЕПЦИЯ.....</b>	<b>7</b>
1.1 ОПИСАНИЕ АЛГОРИТМА В ВИДЕ «ТРИАД».....	7
1.2 ПРИНЦИПЫ ПОСТРОЕНИЯ ПРОЦЕССОРА .....	10
<b>2 АРХИТЕКТУРА .....</b>	<b>12</b>
2.1 СХЕМА МУЛЬТИКЛЕТОЧНОГО ПРОЦЕССОРА.....	12
2.2 ИСПОЛНЕНИЕ ПРОГРАММЫ .....	13
2.3 АРХИТЕКТУРНЫЕ ОСОБЕННОСТИ .....	15
<b>ЗАКЛЮЧЕНИЕ.....</b>	<b>19</b>
<b>ПРИЛОЖЕНИЕ 1 .....</b>	<b>20</b>

## ВВЕДЕНИЕ

«Фон-неймановская» эпоха в компьютерной индустрии завершается. В International Technology Roadmap for Semiconductors (ITRS) отмечается, что эволюционное развитие, доминирующей последние шестьдесят лет на рынке фон-неймановской модели процессора, сходит на нет [<http://itrs.net/Links/2005ITRS/SysDrivers2005.pdf>].

Ретроспективный анализ этого пути показывает, что каждый очередной шаг в совершенствовании фон-неймановской архитектуры требовал все больше усилий и давал все меньшую отдачу.

Учитывая роль архитектуры, создавшуюся ситуацию можно рассматривать как очередную точку бифуркации в развитии компьютерной индустрии. Выход из нее – это поиск качественно нового, пост-неймановского направления развития процессорных архитектур.

На данный момент, в качестве основного архитектурного направления, ведущими производителями процессоров предлагается многоядерность. Но, это решение не может рассматриваться как начало пост-неймановской эпохи. Это экстенсивное и поэтому временное направление развития все той же фон-неймановской модели. Многоядерность не является качественно новым шагом и не решает существующих проблем компьютерной индустрии.

Начиная с первых вычислительных машин, основной тенденцией развития фон-неймановской архитектуры было увеличение уровня параллелизма при выполнении потока команд. Это стремление неразрывно связано с попытками ослабить или обойти ключевой принцип фон-неймановской архитектуры, а именно, упорядоченное, последовательное размещение команд в программе и их исполнение в порядке размещения. Так, например, конвейер – это частичное совмещение исполнения нескольких команд во времени. Суперскалярная организация процессоров и VLIW-процессоры – это совмещение исполнения нескольких команд не только во времени, но и в пространстве.

Требование упорядоченного размещения и исполнения команд – необходимое условие реализации опосредованной формы информационных связей между командами, которая используется в фон-неймановской архитектуре. А именно, результат выполнения любой очередной команды *отчуждается*, т.е. записывается в общедоступную память машины (регистры, ЗУ), и только после этого он доступен (виден) программисту и может использоваться им в качестве операнда для последующих команд.

Наиболее известные попытки уйти от опосредованной формы и, таким образом, обеспечить «естественную» реализацию параллелизма – это потоковые и редуccionные машины, использующие не опосредованное, а явное задание информационных связей между командами.

Так, в потоковой машине адрес команды, потребителя результата выполнения другой команды, задается непосредственно в командном слове команды-источника этого результата. После выполнения команды-источника результат записывается непосредственно в поле операнда команды-потребителя и становится ее частью.

В редуccionной машине адрес команды-источника задается в командном слове команды-потребителя, что также обеспечивает непосредственную передачу и использование результата.

Информационные связи в обеих машинах определяют порядок исполнения. В итоге, оно становится неупорядоченным – «по готовности» или «по запросу». Это обстоятельство, противоречит модели вычислений используемой в наиболее распространенных императивных языках высокого уровня.

Как известно, модель вычислений в императивных языках высокого уровня – это выполнение упорядоченной последовательности операторов. Каждый оператор представляет собой неделимую и целостную языковую конструкцию, описывающую процесс преобразования данных. Порядок выполнения операций внутри оператора задается путем их ранжирования и расстановки скобок, т.е. указанием информационных связей между операциями. Промежуточные результаты

вычислений внутри оператора не отчуждаются и программисту не видны. Отчуждается и виден только результат выполнения оператора. Следовательно, для абстрактной машины, непосредственно реализующий некоторый язык высокого уровня, оператор языка является командой.

Исходное множество операций любого алгоритмического языка изначально зафиксировано и конечно. Множество операторов, которые теоретически могут быть сконструированы с использованием данных операций — потенциально бесконечно и, соответственно, машина с архитектурой, непосредственно реализующей язык высокого уровня, не имеет фиксированной системы команд.

Подобная архитектура лежит в основе принципиально нового направления построения процессоров — мультиклеточных процессоров, качественные и количественные характеристики которых позволяют говорить о появлении нового пост-неймановского поколения и которые были подтверждены разработкой первого мультиклеточного процессора на кристалле МСр0411100101.

# 1 КОНЦЕПЦИЯ

## 1.1 Описание алгоритма в виде «триад»

Любую формулу, например, приведенную на рисунке 1.1(a), можно представить в виде ярусно-параллельной формы так, как показано на рисунке 1.1(б).

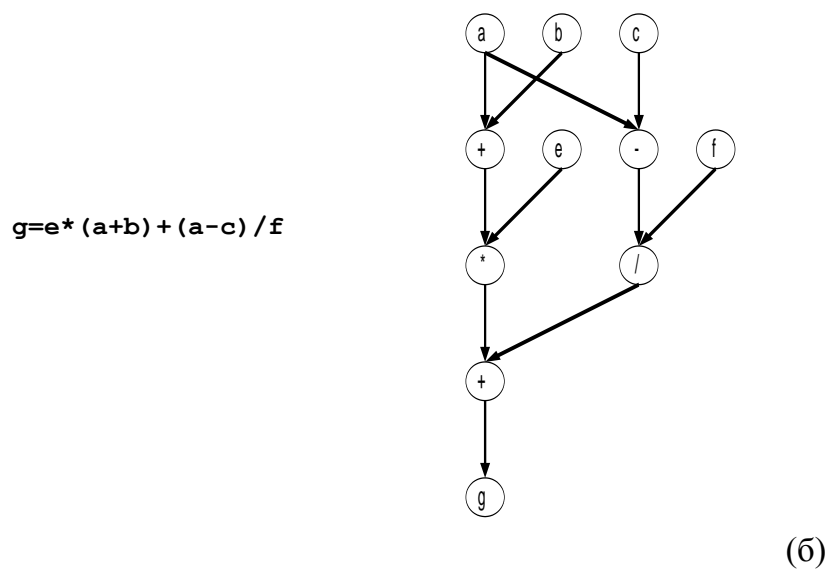


Рисунок 1.1 - Ярусно-параллельное представление формулы: формула(a);  
ярусно-параллельная форма(б).

Пронумеруем последовательно все узлы ярусно-параллельной формы приведенной на рисунке 1.1(б) каким-либо образом, например, слева направо и сверху вниз. Идентификаторы переменных, если их значения используются другими операциями, будем рассматривать как операции чтения данных переменных. Если идентификатор фиксирует результат выполнения операции, то его будем рассматривать как операцию записи данной переменной. Запишем выполняемые операции в порядке номеров. В качестве операндов выполняемой операции укажем номера операций, результаты исполнения которых являются ее аргументами.

Номер	ос	op1	op2
0	RD	a	
1	RD	b	
2	RD	c	
3	+	0	1
4	RD	e	
5	-	0	2
6	RD	f	
7	*	3	4
8	/	5	6
9	+	7	8
10	WR	9	g

Рисунок 1.2 - Описание ЯПФ в виде «триад»

Данное описание, показанное на рисунке 1.2, аналогично промежуточному представлению программы в виде «триад», используемому в процессе компиляции программ, написанных на языках высокого уровня. От классической формы этого представления оно отличается использованием операций чтения и записи и ссылок на эти операции вместо непосредственного использования идентификаторов (ссылок на таблицу идентификаторов). Таким образом, его можно рассматривать как машинно-адаптированную форму исходной программы написанной на языке высокого уровня.

Программа в этом виде представляется пронумерованной последовательностью триад, которая делится на участки. Каждый участок соответствует, как правило, одному оператору программы и содержит подмножество триад, реализующее этот оператор. Очередность записи участков



соответствует очередности записи операторов в программе. Каждая триада описывает выполнение некоторой операции над операндами, заданными идентификаторами или ссылками. Ссылка является номером триады, результат выполнения которой используется в качестве операнда, т.е. ссылка явно задает информационную связь между операциями. При этом результаты триад передаваемые по ссылке не отчуждаются.

Так как информационный обмен между операторами опосредован и осуществляется через отчуждение результатов, то подмножество триад реализующих оператор замкнуто. Оно не имеет ссылок на триады других операторов и триады других операторов не ссылаются на триады данного подмножества. Вне своего подмножества конкретная триада смысла не имеет и, следовательно, не может считаться командой, т.е. информационным сообщением, определяющим действия исполнительного устройства и обладающим целостностью и неделимостью (см. приложение 1). Триада обладает неделимостью, но не обладает целостностью.

Любая триада имеет смысл и может быть выполнена только в определенном контексте. А именно, только после выполнения триад, результаты которых она использует и до тех триад, которые используют ее результаты, т.е. только в составе своего подмножества. Следовательно, программа образованная последовательностью триад является контекстно-зависимой.

Для сравнения, выполнение любой команды в фон-неймановском или потоковом процессоре не зависит от контекста. Программа данных процессоров — контекстно-свободна. Все команды этих процессоров обладают неделимостью и целостностью. Группа команд реализующая оператор обладает целостностью, но не обладает неделимостью.

В общем случае триады оператора могут быть размещены на участке произвольным образом. Их размещение, в отличие от фон-неймановской архитектуры, не определяет порядок исполнения. Очередность их исполнения определяется, как уже отмечалось, информационными связями.

Таким образом, если программа в фон-неймановском процессоре однозначно определяет «что» и «как» надо сделать, то программа, представленная в виде триад однозначно определяет только «что» надо сделать. «Как» надо сделать — решается процессором. И, именно это обстоятельство, дает необходимую степень свободы, которая обеспечивает получение качественно новых и улучшение количественных характеристик процессора.

## **1.2 Принципы построения процессора**

Очевидно, что архитектура процессора способного выполнить программный текст на языке «триад» должна принципиально отличаться от всех известных фон-неймановской и не-фон-неймановских (потокковая и редуционная) архитектур.

Так, во-первых, отличен способ описания информационных связей между операциями и, следовательно, будет отличен способ их реализации. Если в фон-неймановской модели информационные связи между командами (операциями) явно не описываются и реализуются опосредованно, через память (регистры общего назначения, ЗУ), то на языке «триад» они задаются явно, указанием информационных связей между командами. При этом, в отличие от не-фон-неймановских моделей способ задания носит не адресный, а выборочный характер. Результат команды не посылается конкретному потребителю (потокковые процессоры) и не указывается конкретная команда для получения результата (редуционные процессоры), а потребители сами должны выбирать требуемые им результаты из общего потока результатов, который формируется не по заявкам, а императивно, путем выборки и исполнения всех команд линейного участка. Следовательно, архитектура процессора должна иметь механизм идентификации получаемых результатов и интеллектуальную коммутационную среду, обеспечивающую не только широковещательную рассылку всех результатов, но и отбор необходимых результатов для конкретных операций.

Во-вторых, отличен и сам этот процесс. Если потокковые и редуционные процессоры имеют неупорядоченную выборку и исполнение команд, «по готовности» данных или «по запросу» результата, соответственно, то язык «триад»

предполагает последовательную выборку команд линейного участка и исполнение их не только «по готовности» данных, но и «по готовности» потребителей результатов. А именно, выбранная команда не может быть выполнена до тех пор пока не будут получены все операнды (готовность данных) и пока не будут выбраны все команды использующие ее результат (готовность потребителей). Такой подход к исполнению последовательно выбираемых команд связан с их неупорядоченностью.

Неопределенность временного интервала с момента выборки и до момента исполнения команды предполагает использование механизмов буферизации команд, обеспечивающих хранение выбранной команды, комплектование ее операндами, отобранными из потока результатов и выдачу ее на исполнение после выборки всех потребителей ее результата.

## 2 АРХИТЕКТУРА

### 2.1 Схема мультиклеточного процессора

Рассмотрим параллельную систему, показанную на рисунке 2.1 и состоящую из  $N$  процессорных блоков  $PU_0, PU_1, \dots, PU_{n-1}$  связанных между собой однонаправленным коммутатором (SB) типа «каждый с каждым», имеющим  $N$  информационных входов и  $2N$  информационных выходов, а также  $2N$  адресных входов.

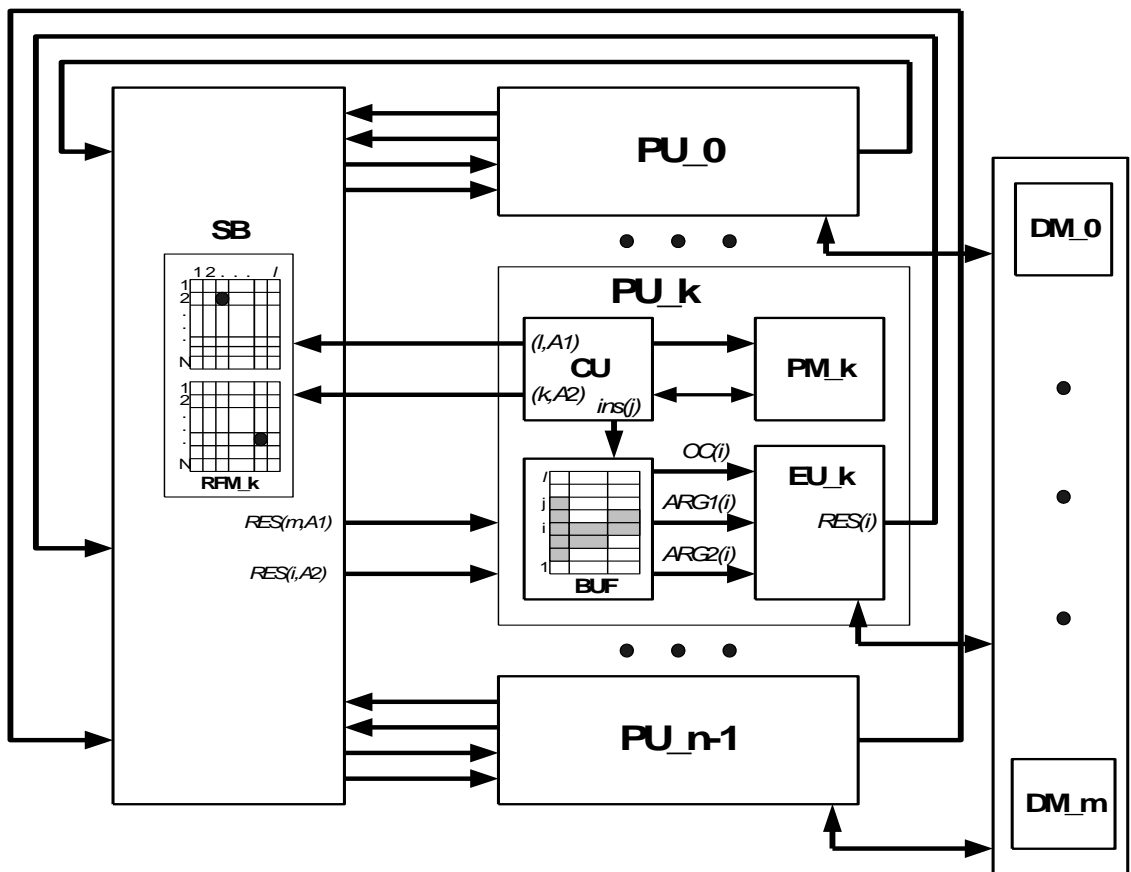


Рисунок 2.1 — Концептуальная схема процессора

Предположим, что система содержит четыре процессорных блока (клетки). Разместим рассматриваемую последовательность команд в  $PM$  процессорных

блоков, начиная с нулевого PU так, как показано на рисунке 2.2. Каждой триаде сопоставим индивидуальный тег (адреса и значения тегов приведены для 0-го PU).

Адрес	Тег	PM_0	PM_1	PM_2	PM_3
A+0	t+0	RD a	RD b	RD c	+ 0,1
A+4	t+4	RD e	- 0,2	RD f	* 3,4
A+8	t+8	/ 5,6	+ 7,8	WR 9,a	

Рисунок 2.2 - Размещение программы в памяти программ процессорных блоков

Для того, чтобы обеспечить параллельное выполнение данной программы функциональными блоками процессора, структура которого приведена на рисунке 2.1, необходимо:

обеспечить согласованную (когерентную) выборку команд находящихся в одной строке;

организовать процесс динамического формирования значений тегов таким образом, чтобы он учитывал количество функциональных блоков осуществляющих выборку команд и их относительные номера.

## 2.2 Исполнение программы

Процесс выборки командных слов инициализируется передачей управления на линейный участок. Он начинается с выборки первой команды линейного участка из памяти программ (PM) и продолжается до выборки последней команды. При этом выполняются следующие действия.

Для каждой выбранной команды динамически формируется значение тега. Оно равно сумме последнего использованного значения тега при выборке команд

и количества функциональных блоков. Значение тега ссылки рассчитывается как сумма начального значения тега на данном линейном участке и номера ссылки. Значение тега в процессе выборки команд изменяется циклически. Его максимальная величина определяется емкостью буфера.

Команда записывается в свободную строку буфера. Если команда содержит значение аргумента непосредственно в командном слове, например, адрес переменной находящейся в памяти данных (DM), то это значение также переписывается в соответствующее поле этой строки буфера. После записи “заготовки” команды в строку буфера устройство управления (CU) приступает к выборке следующей команды.

Процесс приостанавливается после выборки, отмеченной специальным признаком, последней команды линейного участка, которая выбирается и выполняется на общих основаниях. Следует отметить, что последней командой может быть любая команда, а не только команда, которая формирует начальный адрес нового линейного участка. Команда, формирующая начальный адрес следующего участка, может занимать любое место в выполняемом участке. Сформированный ею адрес рассылается всем функциональным блокам, которые на основании его вычисляют свой адрес передачи управления на следующий участок и выполняют эту передачу (возобновляют выборку) после получения сигнала о выборке последней команды текущего участка. Выборка также заканчивается, когда заполнен буфер и возобновляется, когда буфер освобождается.

Поля аргументов в строках буфера организованы как два массива с ассоциативной адресацией. Ассоциативным адресом, по которому осуществляется запись значения поступающего результата при совпадении его с тегом результата является тег запрашиваемого результата.

Команда находится в буфере до тех пор, пока в буфер не придут запрошенные ею результаты и пока в буфера не будут записаны все команды использующие ее результат. Если эти условия удовлетворяются хотя бы для

одной команды, инициализируется процесс исполнения команд. При этом, если готовых к исполнению команд несколько, то на исполнение передается команда, которая была выбрана раньше всех.

Исполнительное устройство (EU) выполняет команду и выдает в буфер её результат с тегом, равным тегу исполненной команды. Завершается или приостанавливается этот процесс тогда, когда в буфере нет готовых к исполнению команд.

Следует отметить, что в процессе исполнения команд, клетки свои действия не согласовывают и работают независимо. Клетка выполняя команду не знает, кто будет потребителем результата. Не определена и очередность исполнения команд. Она определяется потоками данных и команд.

## **2.3 Архитектурные особенности**

1. От фон-неймановской модели мультиклеточная архитектура отличается непосредственным указанием информационных связей между операциями и, соответственно, снятием требования упорядоченного размещения описаний операций в программе.

Эта неупорядоченность делает ненужными все те методы (суперскалярность, широкое командное слово, суперконвейер, предсказание переходов и т.п.), которые, обеспечивая быстродействие, резко усложняли процессы проектирования процессора и инструментальных программных средств (компиляторы, отладчики) и увеличивали их стоимость.

2. От известных не-фон-неймановских архитектур она отличается последовательным способом выборки команд, который обеспечивает реализацию императивных языков программирования, а также использованием для указания информационных связей не адресов команд, а значений динамически формируемых тегов. Команда исполняется по «готовности данных» и «готовности потребителей ее результата».

**3.** Система команд клетки, основана на промежуточном представлении компилируемой программы после синтаксического анализа (триадах) и, фактически, является аппаратной реализацией входного языка программирования. Она минимизирует трудозатраты на создание компиляторов, так как с ее использованием исчезают блоки машинно-ориентированной оптимизации, распараллеливания, резко сокращается объем блока генерации команд. Исчезает понятие «программирование на ассемблере», поскольку язык процессора не наглядный и поэтому «не программируемый». Программное обеспечение становится реально машинно-независимым.

**4.** Неупорядоченность триад обеспечивает, при необходимости, получение после каждой компиляции индивидуального объектного кода для каждого процессора. Это, а также замкнутость подмножеств триад, резко ограничивают возможности незаметного и несанкционированного вмешательства извне в работу системного программного обеспечения.

**5.** Индивидуальность системного кода и использование непривилегированным пользователем для программирования только языка высокого уровня позволяют создать новый и эффективный инструментарий для борьбы с вирусами.

**6.** Триады обеспечивают возможность одновременного чтения и исполнения нескольких команд без анализа их очередности выполнения и информационной связности, т.е. обеспечивают «естественную» реализацию параллелизма. «Естественность» изначально обусловлена видом и механизмами исполнения команд. В мультиклеточном процессоре нет аппаратных средств обеспечивающих выявление информационных связей между выбранными операциями (командами) и распределение их по функциональным устройствам, т.е. нет динамического распараллеливания. Нет и статического распараллеливания, т.к. программа в виде триад хотя и описывает информационные связи, но имеет линейную форму и не содержит каких-либо указаний, что и как можно выполнять параллельно.



7. Полносвязная интеллектуальная коммутационная среда, работающая в режиме «широковещательной» рассылки, не вносит каких-либо топологических ограничений на межклеточный обмен данными и, следовательно, обеспечивает эффективную реализацию любого класса задач (универсальность архитектуры), а также эффективное масштабирование процессора. При увеличении количества клеток и при наличии потенциального параллелизма алгоритма, рост производительности процессора практически равен увеличению количества клеток.

8. Откомпилированная программа может быть выполнена на любом количестве клеток. При этом возможно динамическое изменение их количества, что обеспечивает реализацию методологии постепенной деградации процессора при отказах его клеток. Процессор может перестраиваться и быть работоспособным до тех пор, пока исправна хотя бы одна клетка и коммутационная среда.

Подобная независимость кода от используемых ресурсов создает основу решения проблем непрерывной самоадаптации процессора к потоку задач, а также его самовосстановления после сбоев или после подключения новых ресурсов.

9. Асинхронная и децентрализованная организация мультিকлеточного процессора, как на системном уровне – между клетками (при реализации параллелизма), так и на внутриклеточном уровне – между блоками клетки (при реализации команд), дополнительно обеспечивает:

- минимизацию номенклатуры объектов проектирования и уменьшение их сложности;
- уменьшение площади кристалла, так как объем оборудования при децентрализованном управлении меньше, чем при централизованном;

- увеличение производительности и сокращение энергопотребления в несколько раз, так как позволяет реализовать эффективный вычислительный процесс;
- при реализации, в перспективе, на одном кристалле десятков и сотен клеток, использование индивидуальной системы синхронизации для каждой клетки.

В результате, получается хорошо структурированная и модульная система, позволяющая резко уменьшить сложность процессора и, соответственно, снизить трудозатраты и повысить качество проектирования.

## **ЗАКЛЮЧЕНИЕ**

**Разноплановость и качественный состав преимуществ предлагаемой архитектуры, позволяют позиционировать ее как принципиально новое и высокоэффективное пост-неймановское направление развития микропроцессорной техники.**

# КЛАССИФИКАЦИЯ ПРОГРАММНО-УПРАВЛЯЕМЫХ СИСТЕМ

## 1. Построение и классификация процессорных архитектур

### 1.1. Постановка задачи

Как известно, основой абсолютно всех систем классификации является абстрагирование. Оно позволяет создать модель классифицируемой системы опираясь на главное – параметры классификации, опустив при этом второстепенные детали. Выбор параметров определяется целями классификации и налагаемыми требованиями. Так, в нашем случае обязательным условием является абстрагирование от всех особенностей реализации. Из известных классификаций вычислительных систем [3], этому условию удовлетворяет, и то только частично, классификация Флинна [4]. Она рассматривает вычислительную систему на максимально возможном, но содержательном, уровне абстрагирования, который можно определить как концептуальный. На этом уровне система состоит из командных устройств (устройств управления), исполнительных устройств и устройств памяти, связанных потоками команд и данных. Параметрами классификации являются количественные оценки потоков команд и данных.

Среди потоков, связывающих устройства системы в единое целое, есть те, которые присутствуют абсолютно во всех известных системах и, про которые можно сказать, что они отражают внутреннюю сущность вычислительных систем. Другие – порождены реализацией, т.е. теми техническими решениями, которые были использованы для достижения целей, поставленных при создании конкретных систем.

К числу первых, безусловно относятся потоки команд, формируемые устройствами управления и поступающие в исполнительные устройства. Среди потоков разнообразных данных, циркулирующих в организационно различных системах, общими для всех являются только потоки результатов, формируемые исполнительными устройствами при выполнении ими принятого поток команд. Потоки данных в том понимании, в котором они используются в классификации Флинна, а именно вызываемые («called») при выполнении команд, т.е. поступающие из памяти на обработку, присутствуют только у части систем.

Так, например, потоковая машина не имеет потоков данных, поступающих из памяти и используемых для выполнения очередной команды. Все данные, необходимые для исполнения команды, поступают вместе с ней, как составная часть командного слова.

Очевидно, что в основе как моделей систем, абстрагированных от особенностей реализации, так и в основе их классификации, использующей в качестве параметров классификации оценку потоков команд и данных, должны лежать только те потоки, которые присутствуют абсолютно во всех вычислительных системах.

Отмеченное ранее концептуальное отличие архитектуры потоковых машин от традиционной фон-неймановской архитектуры, связанное с отсутствием вызываемых потоков данных, обеспечивается записью результатов выполненных команд непосредственно в поля аргументов тех команд, которые их используют. Команды исполняются после получения всех необходимых аргументов. Таким образом, очередность их исполнения и, как следствие, вид текущего выходного потока данных непосредственно зависит от потока данных, ранее сформированного исполнительным устройством.

Указанная зависимость имеет ключевое значение для выделения потоковых машин в отдельный вид. Применение только количественной оценки потоков команд и данных, без учета существующих зависимостей между потоками, в принципе, не может решить задачу систематизации подобных (нетрадиционных) архитектур.

Более того, сам подход к формированию этой оценки, используемый в классификации Флинна, не позволяет четко классифицировать даже системы, построенные на базе традиционных, фон-неймановских решений (например – векторно-конвейерные машины).

Так, корректность любой количественной оценки обеспечивается измерительным инструментом, не зависящим от объекта измерения и равенством условий измерения. Применительно к количественной оценке потоков эти два требования можно сформулировать следующим образом:

- на принятом уровне абстрагирования систем должны быть однозначно определены понятия «одиночный» и «множественный», действительные по отношению к любым потокам (команд или данных) и для любых архитектур;
- все систематизируемые вычислительные системы должны рассматриваться на одном уровне абстрагирования.

Ни первое, ни второе требования в классификации Флинна не выполняются. Суть количественных показателей не определена. Их значения устанавливаются постфактум, путем сравнения классифицируемой системы с эталонными для каждого класса образцами и, таким образом, определения ее класса.

В результате, одна и та же характеристика «множественный поток данных», в эталонных образцах классов MIMD (многопроцессорная система) и SIMD (матричный процессор), используется для описания двух принципиально разных явлений. Множества независимых потоков данных в многопроцессорной системе и потока векторных (многокомпонентных) данных в матричном процессоре.

Следует также отметить, что эталонный образец класса SIMD не соответствует концептуальному уровню. Модель функционирования матричного процессора, предусматривающая одновременное выполнение команды над векторным элементом потока данных, отражает одну из возможных реализаций векторных команд. Для других команд, например, скалярных – эта модель не применима. Следовательно, она не применима и на концептуальном уровне, на котором команда рассматривается в обобщенном виде.

Таким образом, для построения множества моделей вычислительных систем, абстрагированных от особенностей реализации, и последующей классификации этого множества необходимо:

1. в качестве потоков данных рассматривать потоки результатов выполнения команд, формируемые исполнительными устройствами;

2. ввести в качестве параметра классификации наличие функциональной зависимости между потоками;

3. однозначно определить понятия «одиночный» и «множественный», действительные по отношению к любым потокам (команд или данных) и для любых архитектур.

Выполнение указанных требований приводит, как показано ниже, к новой системе классификации, предметом которой является множество процессорных архитектур, рассматриваемых на концептуальном уровне.

## 1.2. Основные понятия и определения

На принятом уровне абстрагирования любой процессор будет состоять из устройств управления и исполнительных устройств, связанных в единое целое потоками команд и данных.

Под устройством управления (УУ) понимается источник потока команд. Каждая команда является неделимым и целостным информационным сообщением, однозначно определяющим некоторую неделимую и целостную последовательность действий приемника (приемников). Команда не может быть выдана и исполнена по частям.

Исполнительное устройство (ИУ) – это источник потока промежуточных (не фиксируемых) данных и конечных (фиксируемых) результатов. ИУ принимает, полностью или частично, поступающий ему поток команд и исполняет его. В процессе исполнения ИУ изменяет состояние процессора и/или порождает поток данных (информационных сообщений). Результатом выполнения команды является наблюдаемое (фиксируемое) изменение состояния процессора и/или изменение окружающей среды.

Исполнение последовательности команд — это последовательный переход процессора из одного наблюдаемого состояния в другое сопровождаемое, как правило, последовательным изменением окружающей среды. При этом, процесс исполнения команды аналогичен переходному процессу. Истинное значение параметр (состояние процессора) принимает тогда, когда переходный процесс закончился.

Все источники независимы, а именно процессы, протекающие в одном, непосредственно не связаны с одновременно протекающими процессами в других источниках. Независимость источников не исключает их согласованных действий. Независимы только внутренние процессы источников, но сами действия, например, моменты их начала или окончания могут согласовываться.

Любой источник формирует только один (одиночный) поток, который может поступать в несколько приемников.

Два и более источников одного типа (УУ или ИУ) образуют множество источников. Формируемое ими множество потоков может использоваться как совокупность одиночных потоков, либо как единое целое – множественный поток. Как и одиночный, он также может поступать в несколько приемников.

Понятие «поток» – ключевое. Под потоком понимается непосредственная информационная связь между источником потока и его приемником, объединяющая их и только их в единое целое. Элементы потока существуют только в процессе передачи. Поступая в приемник, они либо используются им, либо исчезают. Для сохранения и последующего использования элемент потока должен быть отчужден, т.е. изъят из потока и размещен вне источника и приемника в окружающей их общей среде (в данном случае, в памяти). Очевидно, что отчуждение разрывает непосредственную связь (поток) между источником и приемником. Такая связь становится опосредованной.

Следует подчеркнуть информационный характер связи. Данная связь существует тогда и только тогда, когда для приемника принципиально важно содержимое элементов входящего потока. Когда именно содержимое, а не очередность элементов потока, например, определяет выполняемые им действия или их результат.

Опираясь на эти понятия можно дать следующие определения архитектуры процессора и вычислительной системы.

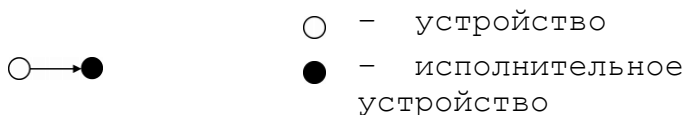
Архитектура процессора – это абстрактная структура, отражающая организацию процессора и представленная в виде ориентированного графа, состоящего из множества вершин, содержащего не менее одного УУ и не менее одного ИУ, а также множества дуг, образованных потоками команд, исходящими из УУ, и потоками данных, исходящими из ИУ, и при этом:

- соответствующий ему неориентированный граф – связный;
- каждый исходящий поток команд является входящим хотя бы для одного ИУ;
- каждое ИУ имеет хотя бы один входящий поток команд.

Архитектура вычислительной системы – это абстрактная структура, отражающая организацию системы и образованная несвязным множеством ориентированных графов, каждый из которых представляет архитектуру процессора.

Следовательно, обмен данными между процессорами в вычислительной системе может осуществляться только через отчуждение элементов потока данных.

Рассмотрим классический фон-неймановский процессор. Графически архитектура данного процессора представлена на рис.1. Она состоит всего из двух вершин (УУ и ИУ), соединенных потоком команд.



**Рис. 1.** Графическое изображение архитектуры классического фон-неймановского процессора: а) графическое изображение; б) условные обозначения.

Следует отметить, что любая другая непосредственная информационная связь, кроме связи по потоку команд, принципиально нереализуема в рамках фон-неймановской модели процессора, описание и исполнение алгоритмов в которой осуществляется только в виде упорядоченной последовательности операций, изменяющих состояние среды (памяти). Этот способ изначально предполагает

отчуждение каждого полученного результата и, соответственно, потоки данных в этой архитектуре отсутствуют.

Расширив нотацию, предложенную М. Флинном, и используя заглавные буквы для описания источников и строчные для указания потоков, данный вид архитектуры можно описать как SISD(si). В скобках, после описания источников, указываются входящие потоки, т.е. потоки от которых зависит формирование исходящего потока. Таким образом, классический фон-неймановский процессор имеет архитектуру с одним источником команд (SI) и одним источником данных (SD), функционирование которого определяется одиночным потоком команд SD(si).











### 1.3. Классификация процессорных архитектур

Очевидно, что использование для количественной оценки источников и потоков двух значений – «одиночный» и «множественный», ограничивает количество возможных вариантов зависимостей. Двухкомпонентное описание архитектуры предполагает ее гомогенность и симметричность. А именно, в случае множественности каких-либо источников существующая зависимость распространяется на все множество (гомогенность), а при множественности потоков команд и данных их размерности считаются равными.

Рассматривая фон-неймановскую архитектуру как отправную точку можно, опираясь на принцип развития, путем последовательного усложнения зависимостей между потоками и исключения несвязных структур, построить полное множество возможных видов процессорных архитектур. Это множество делится на два основных класса:

- архитектуры с хранимой программой (контекстно-свободной программой), представленные в таблице 1;
- архитектуры с хранимым алгоритмом (контекстно-зависимой программой), приведенные в таблице 2.

**Таблица 1.** Базовые (гомогенные) модели архитектур процессоров с хранимой программой

	Фон-неймановские архитектуры	Не фон-неймановские архитектуры				
SISD	 SISD(si)	 SISD(si,sd)	 SI(sd)SD(si)	 SI(sd)SD(si,sd)		
SIMD	 SIMD(si)	 SIMD(si,sd)	 SIMD(si,md)	 SI(md)MD(si)	 SI(md)MD(si,sd)	 SI(md)MD(si,md)



<b>MISD</b>	 MISD(mi)	 MISD(mi,sd)	 MI(sd)SD(mi)	 MI(sd)SD(mi,sd)		
<b>MIMD</b>			 MIMD(si,md)		 MI(sd)MD(si,md)	
				 MI(md)MD(si)	 MI(md)MD(si)	 MI(md)MD(si,md)
	 MIMD(mi)	 MIMD(mi,sd)	 MIMD(mi,md)	 MI(sd)MD(mi)	 MI(sd)MD(mi,sd)	 MI(sd)MD(mi,md)
				 MI(md)MD(mi)	 MI(md)MD(mi,sd)	 MI(md)MD(mi,md)

Принципиальное отличие данных классов заключается в зависимости потоков команд. Поток команд в архитектурах с хранимой программой может зависеть только от потока данных. В архитектурах с хранимым алгоритмом он всегда зависит от самого себя, или от самого себя и от потока данных.

**Таблица 2.** Базовые (гомогенные) модели архитектур процессоров с хранимым алгоритмом

	<b>Не фон-неймановские архитектуры</b>					
<b>SISD</b>	 SI(si)SD(si)	 SI(si)SD(si,sd)	 SI(si,sd)SD(si)	 SI(si,sd)SD(si,sd)		
<b>SIMD</b>	 SI(si)MD(si)	 SI(si)MD(si,sd)	 SI(si)MD(si,md)	 SI(si,md)MD(si)	 SI(si,md)MD(si,sd)	 SI(si,md)MD(si,md)
<b>MISD</b>	 MI(si)SD(mi)	 MI(si)SD(mi,sd)	 MI(si,sd)SD(mi)	 MI(si,sd)SD(mi,sd)		
	 MI(mi)SD(mi)	 MI(mi)SD(mi,sd)	 MI(mi,sd)SD(mi)	 MI(mi,sd)MD(mi,sd)		
<b>MIMD</b>						 MI(si,sd)MD(si,md)

			MI(si)MD(si,md)	MI(si,md)MD(si)	MI(si,md)MD(si,sd)	MI(si,md)MD(si,md)
				MI(si,sd)MD(mi)	MI(si,sd)MD(mi,sd)	MI(si,sd)MD(mi,md)
MI(si)MD(mi)	MI(si)MD(mi,sd)	MI(si)MD(mi,md)		MI(si,md)MD(mi)	MI(si,md)MD(mi,sd)	MI(si,md)MD(mi,md)
				MI(mi,sd)MD(si)	MI(mi,sd)MD(si,sd)	MI(mi,sd)MD(si,md)
MI(mi)MD(si)	MI(mi)MD(si,sd)	MI(mi)MD(si,md)		MI(mi,md)MD(si)	MI(mi,md)MD(si,sd)	MI(mi,md)MD(si,md)
				MI(mi,sd)MD(mi)	MI(mi,sd)MD(mi,sd)	MI(mi,sd)MD(mi,md)
MI(mi)MD(mi)	MI(mi)MD(mi,sd)	MI(mi)MD(mi,md)		MI(mi,md)MD(mi)	MI(mi,md)MD(mi,sd)	MI(mi,md)MD(mi,md)

Как в первой, так и во второй таблицах есть пустые поля. Они отражают существование систем, т.е. несвязных структур, которые были исключены при построении множеств.

Рассмотрим место наиболее известных типов процессоров в данной классификации.

Определение потока данных, как производного от потока команд объединяет архитектуру процессоров таких машин как, например, векторно-конвейерные и матричные, в один вид с классической фон-неймановской машиной. Дальнейшая их классификация должна проводиться уже внутри вида и опираться на особенности реализации. А именно, по типу операций (скалярные, векторные). Среди векторных машин – по способу реализации векторных операций (векторно-конвейерные, матричные).

Одним из первых шагов в развитии фон-неймановской модели, как известно, было внедрение сопроцессоров. Фактически, это было использование дополнительного исполнительного устройства, например, для параллельного выполнения операций с плавающей запятой. Архитектура такого процессора описывается следующим образом – SIMD(si).

Многоядерный процессор, имеющий несколько источников команд и, например, одно исполнительное устройство обладает архитектурой вида MISD(mi). Если исполнительных устройств несколько, и каждое из них может выполнять команды,

поступающие от любого источника, то архитектура будет иметь следующий вид – MIMD(mi).

Другие модели, например, процессор, управляемый потоком данных, имеет архитектуру вида SISD(si, sd), т.е. один источник потока команд и один источник потока данных, поток которого зависит от поступающего одиночного потока команд и от формируемого им же потока данных. Зависимость потока данных от самого себя – характерный признак управления исполнением команд «по готовности». Например, синьпьютер [5-7] имеет формулу MIMD(si,md).

Следует отметить, что несмотря на достаточно большое разнообразие существующих процессоров, концептуально различных среди них сравнительно мало. И если в первой таблице есть реализованные виды, например, четыре вида фон-неймановских архитектур, потоковые архитектуры, то реально существующих видов из второй таблицы – нет.

#### **1.4. Концепция хранимого алгоритма (контекстно-зависимой программы)**

Модель вычислений в императивных языках высокого уровня – это выполнение упорядоченной последовательности операторов. Каждый оператор представляет собой неделимую и целостную языковую конструкцию, описывающую процесс преобразования данных. Порядок выполнения операций внутри оператора задается путем их ранжирования и расстановки скобок, т.е. указанием информационных связей между операциями. Промежуточные результаты вычислений внутри оператора не отчуждаются. Отчуждается только результат выполнения оператора. Следовательно, для абстрактной машины, непосредственно реализующий некоторый язык высокого уровня, оператор языка является командой.

Действительно, рассмотрим промежуточное представление программы в виде триад, получаемое после первой фазы компиляции (синтаксического анализа). Это представление является машинно-адаптированной формой исходного кода написанного на языке высокого уровня.

Программа в этом виде записывается как пронумерованная последовательность триад. Данная последовательность делится на участки. Каждый участок соответствует одному оператору программы и содержит подмножество триад, реализующее этот оператор. Очередность записи участков соответствует очередности записи операторов в программе. Каждая триада описывает выполнение некоторой операции над операндами, заданными идентификаторами или ссылками. Ссылка является номером триады, результат выполнения которой используется в качестве операнда, т.е. ссылка явно задает информационную связь между операциями. При этом, результаты триад передаваемые по ссылке не отчуждаются.

Так как информационный обмен между операторами опосредован и осуществляется через отчуждение результатов, то подмножество триад реализующих оператор замкнуто. Оно не имеет ссылок на триады других операторов и триады других операторов не ссылаются на триады данного подмножества. Следует отметить, что вне своего подмножества триады смысла не имеют и не могут считаться системой команд.

Они безусловно обладают неделимостью, но не обладают целостностью и, следовательно, не являются командами.

Любая триада имеет смысл и может быть выполнена только в определенном контексте. А именно, только после выполнения триад, результаты которых она использует и до тех триад которые используют ее результаты, т.е только в составе своего подмножества. Таким образом, целостностью и неделимостью обладает все подмножество триад в целом, а не отдельные триады.

Для сравнения, выполнение любой команды в фон-неймановском или потоковом процессоре не зависит от контекста. Все команды этих процессоров обладают неделимостью и целостностью. Группа команд, реализующая оператор обладает целостностью, но не обладает неделимостью.

Как известно, исходное множество операций любого алгоритмического языка изначально зафиксировано и конечно. Множество операторов, которые теоретически могут быть сконструированы с использованием данных операций — потенциально бесконечно. Так как каждая программа имеет свой набор операторов и, соответственно, свою систему команд, то можно сказать, что машина, непосредственно реализующая язык высокого уровня, не имеет фиксированной системы команд.

В общем случае триады оператора могут быть размещены на участке произвольным образом. Последовательность выдачи их на исполнение определяется информационными связями, т.е. предшествующими (формирующими операнды) и последующими (использующими результат) триадами. Так как команда однозначно определяется последовательностью задаваемых ею операций, то можно сказать, что исполняемая команда формируется непосредственно в процессе работы процессора и ее вид зависит от самой себя, а именно от ее внутренней организации. Соответственно и поток команд зависит от самого себя.

Зависимость потока команд от самого себя не единственное отличие класса архитектур с хранимым алгоритмом (контекстно-зависимой программой) от класса архитектур с хранимой программой (контекстно-свободной программой). Можно также указать еще на два фундаментальных различия данных классов.

Первое – множество программ любой архитектуры с контекстно-свободной программой счетно. Множество программ любой архитектуры с контекстно-зависимой программой имеет мощность континуума.

Второе – архитектуры с контекстно-зависимой программой в общем случае не могут быть представлены машиной Тьюринга.

## 2. Мультиклеточный процессор

Первым параллельным процессором из класса архитектур с хранимым алгоритмом является создаваемый в настоящее время мультиклеточный процессор вида  $MI(mi)MD(mi,md)$ . Он образован множеством независимо, но согласованно, функционирующих клеток, связанных коммутационной средой.

Машинный язык процессора является развитием языка триад.

Текст программы на языке триад не связан каким-либо образом с количеством клеток. Эта «ресурсная» независимость, неупорядоченность команд внутри их контекстной последовательности (линейного участка) и рассылка всем клеткам всех

получаемых результатов обеспечивают «естественную» реализацию параллелизма (без решения задачи распараллеливания), а также эффективное масштабирование процессора [8].

Указанные особенности также делают ненужными все те методы (суперскалярность, широкое командное слово, суперконвейер, спекулятивное и предикатное исполнение и т.п.), которые обеспечивая быстрое действие фон-неймановской модели, резко усложняли ее организацию. Отказ от этих методов и использование децентрализованной организации позволяет резко уменьшить сложность мультиклеточного процессора и, соответственно, снизить трудозатраты и повысить качество проектирования.

Одновременно с этим, по сравнению с традиционными фон-неймановскими решениями, улучшаются и характеристики процессора. Первые оценки позволяют говорить о росте производительности в 2-4 раза и снижении энергопотребления в 10 – 15 раз.

Так как мультиклеточный процессор является аппаратной реализацией входного языка программирования, опирающейся не на внешнюю форму, а на сущность языковых выражений, то:

- сохраняется все программное обеспечение, созданное на традиционных императивных языках высокого уровня, и повышается его эффективность;
- процесс компиляции с языка высокого уровня, фактически, ограничивается начальной машинно-независимой («front-end») фазой, что позволяет резко сократить затраты на разработку компиляторов;
- исчезает понятие «программирование на ассемблере», поскольку язык процессора не нагляден и поэтому практически «не программируемый»;
- наиболее эффективной формой существования программ становится исходный текст и, соответственно, программы становятся открытыми.

Следует отметить, что ни одна, из ранее созданных процессорных архитектур, не обеспечивает комплексного решения тех проблем, которые решаются мультиклеточным процессором. Это позволяет говорить о качественно новом направлении в построении микропроцессоров.

### **3. Заключение**

Предложенная система классификации программно-управляемых систем позволяет систематизировать известные процессорные архитектуры и целенаправленно подойти к выбору нового направления их развития.

Таким новым направлением является создание архитектур с хранимым алгоритмом (контекстно-зависимой программой). Данный класс архитектур решает целый ряд задач компьютерной индустрии, которые принципиально не могут быть решены в рамках традиционной фон-неймановской модели или других известных моделей. Это уменьшение сложности и стоимости проектирования процессора, при одновременном повышении его технических характеристик. Повышение языкового уровня процессора и, как следствие, сокращение затрат на создание как компиляторов, так и программного обеспечения в целом. Реализация параллелизма «естественным»

образом при использовании традиционных императивных языков, эффективное динамическое реконfigurирование процессора.

## Список литературы

- Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем. СПб.: Питер, 2004. 668с.
- Винер Н. Кибернетика или управление и связь в животном и машине – М.: Советское радио, 1968. 326с.
- Классификации архитектур вычислительных систем. <http://www.parallel.ru>.
- Flynn M. Very high-speed computing system // Proc. IEEE. 1966. N 54. P.1901-1909.
- Патент № 2179333 RU «Синергическая вычислительная система».
- Патент № 2198422 RU «Асинхронная синергическая вычислительная система».
- 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. P. CD-ROM.
- Стрельцов Н.В. Реализация параллелизма в мультиклеточном процессоре // Труды III Международной научной конференции «Параллельные вычисления и задачи управления». М.: Институт проблем управления, 2-4 октября 2006. С.337-347.