

## Тема 5\_2010: Процессоры.

### 0. Краткая история и основные термины

1. Понятие архитектуры ЭВМ и ВС .....	1
2. RISC- и CISC-архитектуры.....	3
3. RISC архитектура .....	3
3.1. Основные принципы RISC-архитектуры .....	3
3.2. Отличительные черты RISC- и CISC- архитектур .....	5
3.3. Некоторые задачи реализации RISC-процессоров .....	6
3.4. Методы адресации и типы команд.....	10
3.5. Компьютеры со стековой архитектурой.....	12
3.6. Оптимизация системы команд.....	18

## 1. Понятие архитектуры ЭВМ и ВС

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

Детализацией архитектурного и структурного построения ЭВМ занимаются различные категории специалистов вычислительной техники. Инженеры-схемотехники проектируют отдельные технические устройства и разрабатывают методы их сопряжения друг с другом. Системные программисты создают программы управления техническими средствами, информационного взаимодействия между уровнями, организации вычислительного процесса. Программисты-прикладники разрабатывают пакеты программ более высокого уровня, которые обеспечивают взаимодействие пользователей с ЭВМ и необходимый сервис при решении ими своих задач.

Пользователя интересуют обычно более общие вопросы, касающиеся его взаимодействия с ЭВМ (человеко-машинного интерфейса), начиная со следующих групп характеристик ЭВМ, определяющих ее структуру:

- технические и эксплуатационные характеристики ЭВМ (быстродействие и производительность, показатели надежности, достоверности, точности, емкость оперативной и внешней памяти, габаритные размеры, стоимость технических и программных средств, особенности эксплуатации и др.);
- характеристики и состав функциональных модулей базовой конфигурации ЭВМ; возможность расширения состава технических и программных средств; возможность изменения структуры;
- состав программного обеспечения ЭВМ и сервисных услуг (операционная система или среда, пакеты прикладных программ, средства автоматизации программирования).

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

В узком смысле под архитектурой понимают **архитектуру набора команд**. Архитектура набора команд определяет границу между аппаратным и программным обеспечением и представляет ту часть системы, которая видна программисту или разработчику компиляторов.

## **2.RISC- и CISC-архитектуры**

Двумя основными архитектурами набора команд, используемыми компьютерной промышленностью на современном этапе развития вычислительной техники, являются архитектуры CISC и RISC.

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

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

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

Для снятия указанных ограничений в Беркли (США, Калифорния) была разработана RISC(Restricted (reduced) instruction set computer)-архитектура (регистро-ориентированная архитектура). Компьютеры с такой архитектурой иногда называют компьютерами с сокращенным набором команд. Суть ее состоит в выделении наиболее употребительных операций и создании архитектуры, приспособленной для их быстрой реализации. Это позволило в условиях ограниченных ресурсов разработать компьютеры с высокой пропускной способностью.

## **3.RISC архитектура**

### **3.1. Основные принципы RISC-архитектуры**

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

В самом начале 80-х годов почти одновременно завершились теоретические исследования в области RISC-архитектуры, проводившиеся в Калифорнийском, Стэнфордском университетах, а также в лабораториях фирмы IBM. Особую

значимость имел проект RISC-1, который возглавили профессора Давид Паттерсон и Карло Секуин. Именно они ввели в употребление термин RISC и сформулировали **четыре основных принципа RISC-архитектуры**:

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

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

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

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

Развитие RISC-архитектуры в значительной степени определяется успехами в области проектирования оптимизирующих компиляторов. Только современная технология компиляции позволяет эффективно использовать преимущества большого регистрового файла, конвейерной организации и высокой скорости выполнения команд. Есть и другие свойства процесса оптимизации в технологии компиляции, обычно используемые в RISC-процессорах: реализация задержанных

переходов и суперскалярная обработка, позволяющие в один и тот же момент времени посылать на выполнение несколько команд.

### 3.2. Отличительные черты RISC- и CISC- архитектур

Как уже отмечалось выше, двумя основными архитектурами набора команд, используемыми компьютерной промышленностью на современном этапе развития вычислительной техники, являются архитектуры CISC и RISC.

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

Основные черты RISC-архитектуры с аналогичными по характеру чертами CISC-архитектуры отображаются следующим образом (табл. 3.1):

Таблица 3.1. Основные черты архитектуры

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

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

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

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

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

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

### **3.3. Некоторые задачи реализации RISC-процессоров**

#### ***Выбор оптимального набора операций.***

Обычно по некоторой джентльменской смеси задач, для выполнения которой предназначен компьютер, выбирают пакет контрольных программ и строят для них профиль их выполнения либо используют метод статических или динамических измерений параметров самих программ.

При профилировании программы определяется доля общего времени центрального процессора, затрачиваемого на выполнение каждого оператора (операции) программы. Анализ полученных результатов позволит выявить характерные особенности профилируемой программы.

При статических или динамических измерениях подсчитывается, сколько раз в программе встречается тот или иной оператор (операция) или как часто признаки принимают положительные или отрицательные значения в тексте программы (статика) либо в результате выполнения (динамика).

Сочетание результатов, полученных в ходе предложенных исследований, дает общую картину анализируемой программы. Вот результаты одного из таких измерений в статике, проведенные для программ-компиляторов:

- операторы присваивания – 48 %;
- условные операторы – 15; циклы – 16;
- операторы вызова-возврата – 18;
- прочие операторы – 3 %.

Измерение трехсот процедур, используемых в программах – операционных системах в динамике показали следующие измерения типов операндов:

- константы – 33 %;
- скаляры – 42;
- массивы (структуры) – 20
- и прочие – 5 %.

При этом статистика среди команд управления потоком данных следующая. В разных тестовых пакетах программ

- команды условного перехода занимают от 66 до 78 %,
- команды безусловного перехода – от 12 до 18 %,
- частота переходов на выполнение составляет от 10 до 16 %.

Отсюда можно сделать вывод, что операторы присваивания занимают основную часть в программах-компиляторах, а операнды типа константа и локальные скаляры составляют основную часть операндов в процедурах, к которым происходит обращение в процессе выполнения программы.

Подобные количественные и качественные измерения образуют основу для оптимизации процессорной архитектуры.

Так, если операторы присваивания занимают 48 % всех операторов, то ясно, что **операцию доступа к операндам следует реализовать аппаратно**. Особенно это важно, если речь идет о нечисловых задачах, в которых операции вычислительного плана, как правило, просты. В условных операторах, операторах цикла и вызова-возврата производятся манипуляции над множеством операндов, что также подтверждает необходимость **аппаратной реализации операции доступа к операндам**.

При анализе типов операндов обычно учитывают три категории:

- **константы** – не меняются во время выполнения программы и имеют, как правило, небольшие значения;

- **скаляры** – обращение к ним происходит, как правило, явно по их имени. Их обычно немного, и они описываются в процедурах как локальные;

- **обращение к элементам массивов и структур** происходит посредством индексов и указателей, т. е. через косвенную адресацию. Этих элементов, как правило, много.

Для осуществления доступа к операнду необходимо вначале определить физический адрес ячейки, где хранится операнд, а затем осуществить доступ к операнду. Если операнд – константа, ее можно указать в команде. Доступ к ней может быть осуществлен немедленно после обращения. В других случаях все зависит от типа памяти, где хранится операнд. Регистровые блоки и кэш-память мало отличаются по емкости, скорости доступа и стоимости, однако существенно отличаются по накладным расходам на адресацию. Обращение к кэш-памяти осуществляется при помощи адресов полной длины, которые требуют более высокой полосы пропускания каналов обмена. Как правило, этот адрес приходится формировать во время выполнения программы, что как минимум требует полноразрядного сложения или обращения к регистру либо и того, и другого. Регистровые блоки адресуются короткими номерами регистров, которые обычно указываются в команде, что упрощает их декодирование. Однако регистровая память может хранить только определенные типы данных, в то время как кэш-память может использоваться и как основная. В связи с этим распределение скалярных переменных по регистрам необычайно важно, ибо существенно влияет на скорость обработки.

Необходимо решить вопрос с размером регистровых окон, ибо в обычной архитектуре окна однорегистровые, что требует операции запоминания-восстановления при каждом вызове-возврате.

При вызовах процедур необходимо запоминать содержимое регистров, а при возвращении – восстанавливать их, на что тратится значительное время. Вызовы процедур в современных структурированных программах делаются довольно часто, при этом суммарные накладные расходы на выполнение команд вызова и возврата из процедур на стандартных процессорах доходит до 50 % всех обращений к памяти в программе. Чтобы уменьшить время передачи данных между процедурами-родителями и процедурами-дочерьми (в случае, когда глубина их вложенности больше единицы), можно создать блок регистров, предоставив и родителям, и дочерям доступ к некоторым из них. Другими словами, необходимо создать перекрывающиеся блоки регистров. Такая возможность реализуется через перекрывающиеся "окна", накладываемые на блок регистров. Этот механизм реализован в RISC-архитектуре. Многие измерения показывают, что около 97 % процедур имеют не более шести параметров, что требует, чтобы перекрытие соседних окон шло где-то на пять регистров.



Процессор всегда содержит указатель текущего окна с возможностью его модификации в рамках реализованной глубины вложения процедур. Если глубина переполняется, часть содержимого окон направляется в основную память, чтобы иметь место для новых процедур. Для простоты реализации в ОП направляется содержимое целого окна. Это говорит о целесообразности введения многооконного механизма с заданным размером окна. Размер окна определяется количеством передаваемых параметров, а количество окон – допустимой вложенностью процедур.

Среди достоинств применения больших регистровых блоков можно выделить высокую скорость доступа к блокам и увеличение частоты вызова процедур. Среди недостатков – высокая стоимость большого регистрового блока, организуемого в ущерб другим функциональным блокам на кристалле.

Аппаратная реализация механизма перекрытия окон должна вводиться, если получаемый выигрыш перекрывает затраты.

Так как значительная часть процедур требует от трех до шести параметров, то следует подумать об оптимизации использования окон. Во-первых, пересылать в ОП в случае переполнения регистрового блока можно не все окна, а только содержимое регистров, используемых для данной процедуры, и, во-вторых, применять механизм с окнами переменных размеров, определяемых во время выполнения.

Перспективной разновидностью RISC-архитектуры явилась архитектура **SPARC (Scalable Processor Architecture)**. *Scaling – масштабирование, т. е. способность представлять данные таким образом, чтобы и они, и результат проводимых с ними вычислений находились в диапазоне чисел, которые могут обрабатываться в рамках данного процесса или на данном оборудовании.*

Новая серия SPARCSTATIONS фирмы Sun Microsystems базируется на SPARC-архитектуре. Первые модели этой фирмы были изготовлены уже в 1989 г. Операционной средой для всех станций фирмы Sun является SunOS – разновидность OS Unix, снабженная многооконным графическим интерфейсом Open Look.

Для повышения скорости обработки данных используются компьютеры с VLIW (Very Long Instruction Word)-архитектурой. Структура команды таких компьютеров наряду с кодом операции и адресами операндов включает теги и дескрипторы. Наряду с существенным ускорением обработки данных такая архитектура позволяет экономить память при достаточном количестве обращений к командам и сокращать общее количество команд в системе команд.

### 3.4. Методы адресации и типы команд

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

В табл. 3.2 представлены основные методы адресации операндов.

Таблица 3.2. Методы адресации

Метод адресации	Пример команды	Смысл команды	Использование команды
Регистровая	Add R4, R3	$R4 = R4 + R3$	Для записи требуемого значения в регистр
Непосредственная или литерная	Add R4, #3	$R4 = R4 + 3$	Для задания констант
Базовая со смещением	Add R4, 100(R1)	$R4 = R4 + M(100 + R1)$	Для обращения к локальным переменным
Косвенная регистровая	Add R4, (R1)	$R4 = R4 + M(R1)$	Для обращения по указателю к вычисленному адресу
Индексная	Add R3, (R1+R2)	$R3 = R3 + M(R1 + R2)$	Полезна при работе с массивами: R1 – база, R3 – индекс
Прямая или абсолютная	Add R1, (1000)	$R1 = R1 + M(1000)$	Полезна для обращения к статическим данным
Косвенная	Add R1, @(R3)	$R1 = R1 + M(M(R3))$	Если R3 – адрес указателя p, то выбирается значение по этому указателю
Автоинкрементная	Add R1, (R2)+	$R1 = R1 + M(R2)$ $R2 = R2 + d$	Полезна для прохода в цикле по массиву с шагом: R2 – начало массива. В каждом цикле R2 получает приращение d
Автодекрементная	Add R1, (R2)–	$R2 = R2 - d$ $R1 = R1 + M(R2)$	Аналогична предыдущей. Обе могут использоваться для

			<i>реализации стека</i>
<i>Базовая индексная со смещением и масштабированием</i>	<i>Add</i>  <i>R1, 100(R2)(R3)</i>	$R1 = R1 + M(100) + R2 + R3 * d$	<i>Для индексации массивов</i>

Адресация непосредственных данных и литерных констант обычно рассматривается как один из методов адресации памяти (хотя значения данных, к которым в этом случае производятся обращения, являются частью самой команды и обрабатываются в общем потоке команд).

В табл. 3.2 на примере команды сложения (Add) приведены наиболее употребительные названия методов адресации, хотя при описании архитектуры в документации производители компьютеров и ПО используют разные названия для этих методов. В табл. 3.2. знак "=" используется для обозначения оператора присваивания, а буква М обозначает память (Memory). Таким образом М(R1) обозначает содержимое ячейки памяти, адрес которой определяется содержимым регистра R1.

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

Команды традиционного машинного уровня можно разделить на несколько типов, которые показаны в табл. 3.3.

*Таблица 3.3. Основные типы команд*

<b>Тип операции</b>	<b>Примеры</b>
<i>Арифметические и логические</i>	<i>Целочисленные арифметические и логические операции: сложение, вычитание, логическое сложение, логическое умножение и т. д.</i>
<i>Пересылки данных</i>	<i>Операции загрузки/записи</i>
<i>Управление потоком команд</i>	<i>Безусловные и условные переходы, вызовы процедур и возвраты</i>
<i>Системные операции</i>	<i>Системные вызовы, команды управления виртуальной памятью и т. д.</i>
<i>Операции с плавающей точкой</i>	<i>Операции сложения, вычитания, умножения и деления над вещественными числами</i>
<i>Десятичные операции</i>	<i>Десятичное сложение, умножение, преобразование форматов и т. д.</i>
<i>Операции над строками</i>	<i>Пересылки, сравнения и поиск строк</i>

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

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

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

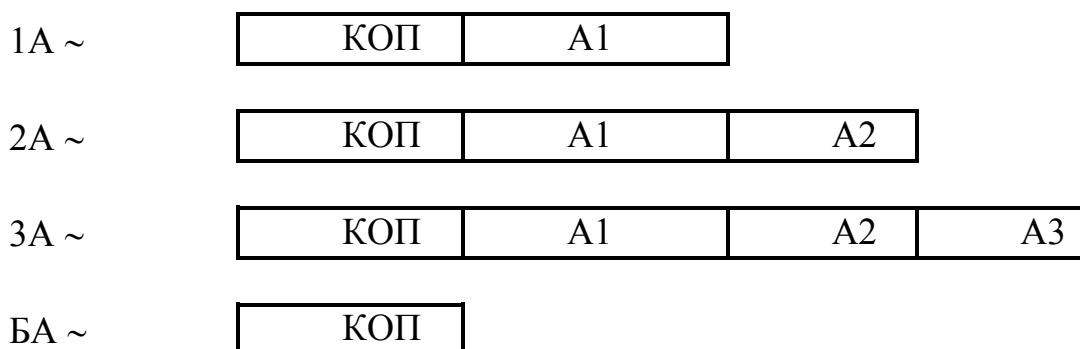
### 3.5. Компьютеры со стековой архитектурой

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

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

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

На сегодняшний день наибольшее распространение получили следующие структуры команд: одноадресные (1A), двухадресные (2A), трехадресные (3A), безадресные (БА), команды с большой длиной слова (VLIW – БДС) (рис. 3.1):



БДС ~

КОП	Адреса	Теги	Дескрипторы
-----	--------	------	-------------

Рис. 3.1. Структуры команд

Причем операнд может указываться как адресом, так и непосредственно в структуре команды.

В случае БА-команд операнды выбираются и результаты помещаются в стек (магазин, гнездо). Типичными первыми представителями БА-компьютеров являются KDF-9 и МВК "Эльбрус". Их характерной особенностью является наличие стековой памяти.

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

Рассмотрим функционирование процессора со стековой организацией памяти.

При выполнении различных вычислительных процедур процессор использует либо новые операнды, до сих пор не выбиравшиеся из памяти компьютера, либо операнды, употреблявшиеся в предыдущих операциях. В процессорах с классической структурой обращение к любому операнду (1А-ЭВМ) требует цикла памяти.

Рассмотрим пример.

Пусть процессор вычисляет значение выражения

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

Программа решения этой задачи для одноадресного компьютера может быть следующей (табл. 3.4).

Таблица 3.4. Пример программы

Номер команды	Команда	Комментарии
1	$c \rightarrow \Sigma$	
2	$(\Sigma) + b$	
3	$(\Sigma) \rightarrow P_1$	$P_1$ – рабочая ячейка
4	$a \rightarrow \Sigma$	

5	$(\sum) \bullet a$	
6	$(\sum) \rightarrow P_2$	$P_2$ – рабочая ячейка
7	$b \rightarrow \sum$	
8	$(\sum) \bullet b$	
9	$(\sum) + (P_2)$	
10	$(\sum) : (P_1)$	$\frac{a^2 + b^2}{b + c} \rightarrow \sum$

*Замечание.* Выполнение команды типа  $(\sum) \otimes (P_i)$  подразумевает, что результат операции помещается в первый регистр, в данном случае в регистр  $\sum$ .

Как следует из приведенной программы, операнд  $a$  выбирается из памяти 2 раза (команды 4 и 5),  $b$  – 3 раза (команды 2, 7 и 8). Кроме того, потребовались дополнительные обращения к памяти для запоминания и вызова из памяти результатов промежуточных вычислений (команды 3, 6, 9, 10).

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

Указанные соображения получили свое воплощение в ряде логических структур процессора. Одна из них – процессор со стековой памятью. Принцип ее работы поясняет схема, представленная на рис. 3.2.

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

Движение вниз:  $(P_1) \rightarrow P_2, (P_2) \rightarrow P_3, \dots$ , а  $P_1$  заполняется данными из главной памяти.

Движение вверх:  $(P_n) \rightarrow P_{n-1}, (P_{n-1}) \rightarrow P_{n-2}$ , а  $P_n$  заполняется нулями.

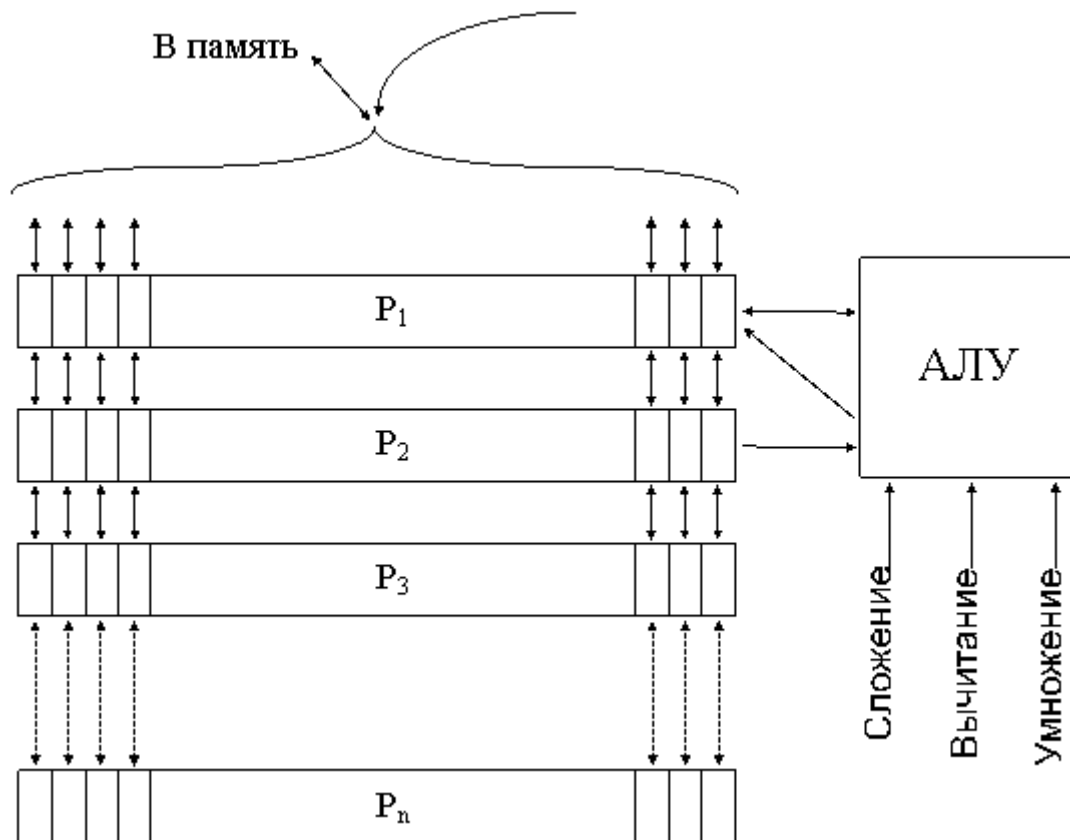


Рис. 3.2. Стековая организация процессора

Регистры  $P_1$  и  $P_2$  связаны с АЛУ, образуя два операнда для выполнения операции. Результат операции записывается в  $P_1$ . Следовательно, АЛУ выполняет операцию  $(P_1) \otimes (P_2) \rightarrow P_1$ .

Одновременно с выполнением арифметической операции (АО) осуществляется продвижение операндов вверх, не затрагивая  $P_1$ , т. е.  $(P_3) \rightarrow P_2$ ,  $(P_4) \rightarrow P_3$  и т. д.

Таким образом, АО используют подразумеваемые адреса, что уменьшает длину команды. В принципе, в команде достаточно иметь только поле, определяющее код операции. Поэтому компьютеры со стековой памятью называют безадресными. В то же время команды, осуществляющие вызов или запоминание информации из главной памяти, требуют указания адреса операнда. Поэтому в ЭВМ со стековой памятью используются команды переменной длины. Например, в KDF-9 команды АО – однослововые, команды обращения к памяти и передач управления – трехслововые, остальные – двуслововые.

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

Для эффективного использования возможностей такой памяти в ЭВМ вводятся спецкоманды:

•дублирование  $\sim (P_1) \rightarrow P_2, (P_2) \rightarrow P_3, \dots$  и т. д., а  $(P_1)$  остается при этом неизменным;

•реверсирование  $\sim (P_1) \rightarrow P_2$ , а  $(P_2) \rightarrow P_1$ , что удобно для выполнения некоторых операций.

Рассмотрим тот же пример для новой ситуации (табл. 3.5):

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



Таблица 3.5. Реализация программы со стековой памятью

№ п/п	Команда	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>
1	Вызов <i>b</i>	<i>b</i>			
2	Дублирование	<i>b</i>	<i>B</i>		
3	Вызов <i>c</i>	<i>c</i>	<i>B</i>	<i>B</i>	
4	Сложение	<i>b+c</i>	<i>B</i>		
5	Реверсирование	<i>b</i>	<i>b+c</i>		
6	Дублирование	<i>b</i>	<i>B</i>	<i>b+c</i>	
7	Умножение	<i>b</i> <sup>2</sup>	<i>b+c</i>		
8	Вызов <i>a</i>	<i>a</i>	<i>b</i> <sup>2</sup>	<i>b+c</i>	
9	Дублирование	<i>a</i>	<i>A</i>	<i>b</i> <sup>2</sup>	<i>b+c</i>
10	Умножение	<i>a</i> <sup>2</sup>	<i>b</i> <sup>2</sup>	<i>b+c</i>	
11	Сложение	<i>a</i> <sup>2</sup> + <i>b</i> <sup>2</sup>	<i>b+c</i>		
12	Деление	$\frac{a^2+b^2}{b+c}$			

Как следует из табл. 3.5, понадобились лишь **три обращения к памяти** для вызова операндов (команды 1, 3, 8). Меньше обращений принципиально невозможно. Операнды и промежуточные результаты поступают для операций в АУ из стековой памяти; 9 команд из 12 являются безадресными.

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

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

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

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

**Недостатки** стековой организации памяти:

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

### 3.6. Оптимизация системы команд

Важным вопросом построения любой системы команд является оптимальное кодирование команд. Оно определяется количеством регистров и применяемых методов адресации, а также сложностью аппаратуры, необходимой для декодирования. Именно поэтому в современных RISC-архитектурах используются *достаточно простые методы адресации, позволяющие резко упростить декодирование команд*. Более сложные и редко встречающиеся в реальных программах методы адресации реализуются с помощью дополнительных команд, что, вообще говоря, приводит к увеличению размера программного кода. Однако такое увеличение программы с лихвой окупается возможностью простого увеличения частоты RISC-процессоров. Этот процесс можно наблюдать сегодня, когда максимальные тактовые частоты практически всех RISC-процессоров (Alpha, R4400, HyperSPARC и Power2) превышают тактовую частоту, достигнутую процессором Pentium.

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

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

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

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

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

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

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

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