Тема 5. Архитектуры процессора

Основные вопросы:

2.1. RISC- и CISC-архитектуры, основные принципы построения и реализации

2.2. Методы адресации и типы машинных команд. Оптимизация системы команд

2.3. Компьютеры со стековой архитектурой

2.4. Микропроцессоры

2.4.1. Основные характеристики

2.4.2. Структура базового микропроцессора. Взаимодействие элементов

2.4.3. Поколения микропроцессоров семейства Intel

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.

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**.

2.1. RISC- и CISC-архитектуры, основные принципы построения и реализации

На современном этапе развития вычислительной техники существуют две основные архитектуры набора команд, используемые компьютерной промышленностью, - архитектуры **CISC** (**C**omplete **I**nstruction **S**et **C**omputer) и **RISC** (**R**estricted (reduced) **I**nstruction **S**et **C**omputer).

Основоположником CISC-архитектуры – архитектуры с полным набором команд 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.

Немного истории:

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 регистро-ориентированная RISC – 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.

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.

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 основной закон RISC был и остается незыблемым: 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.

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 в сравнении с аналогичными по характеру чертами CISC-архитектуры отображаются в табл. 2.1:

Таблица 2.1.

Основные черты архитектуры

CISC-архитектура	**RISC-архитектура**
Длина команды -варьируется	Единая
Расположение полей - варьируется	неизменное
Многобайтовые команды	Однобайтовые команды
Малое количество специализированных регистров	Большое количество регистров общего назначения
Сложные команды	Простые команды
Одна или менее команд за один цикл процессора	Несколько команд за один цикл процессора
Доступ к памяти – как часть команд различных типов	Выполняют только специальные команды
Традиционно одно исполнительное устройство	Несколько исполнительных устройств

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 – ***R*1:=*R*2, *R*3**. 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 Стэндфордского 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.

2.2. Методы адресации и типы машинных команд. Оптимизация системы команд (*самостоятельная проработка*)

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

В табл. 2.2 представлены основные методы адресации операндов.

Таблица 2.2.

Методы адресации

Метод адресации	**Пример команды**	**Смысл команды**	**Использование команды**
Регистровая	*Add R*4, *R*3	*R*4 = *R*4+*R*3	Для записи требуемого значения в регистр
Непосредственная или литерная	*Add R*4, #3	*R*4 = *R*4+3	Для задания констант
Базовая со смещением	*Add R*4, 100*(R*1*)*	*R*4= *R*4+*M(*100+*R*1*)*	Для обращения к локальным переменным
Косвенная регистровая	*Add R*4, *(R*1*)*	*R*4 = *R*4+*M(R*1*)*	Для обращения по указателю к вычисленному адресу
Индексная	*Add R*3, *(R*1+*R*2*)*	*R*3 = *R*3+*M(R*1+*R*2*)*	Полезна при работе с массивами: R1 – база, R3 – индекс
Прямая или абсолютная	*Add R*1, *(*1000*)*	*R*1=*R*1+*M(*1000*)*	Полезна для обращения к статическим данным
Косвенная	*Add R*1, *@(R*3*)*	*R*1 = *R*1+*M(M(R*3*))*	Если *R*3 – адрес указателя *p*, то выбирается

			значение по этому указателю
Автоинкрементная	*Add R*1*, (R*2*)+*	*R*1 = *R*1+*M(R*2*)* *R*2 = *R*2+*d*	Полезна для прохода в цикле по массиву с шагом: *R*2 – начало массива. В каждом цикле *R*2 получает приращение *d*
Автодекрементная	*Add R*1*, (R*2*)–*	*R*2 = *R*2–*d* *R*1 = *R*1+*M(R*2*)*	Аналогична предыдущей; обе могут использоваться для реализации стека
Базовая индексная со смещением и масштабированием	*Add R*1,100*(R*2*)(R*3*)*	*R*1=*R*1+*M(*100*)*+*R*2+ *R*3**d*	Для индексации массивов

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. 2.2 на примере команды сложения (*Add*) приведены наиболее употребительные названия методов адресации, хотя при описании архитектуры в документации производители компьютеров и программного обеспечения используют разные названия для этих методов. В табл. 2.2 знак "=" используется для обозначения оператора присваивания, а буква *M* обозначает память (*Memory*). Таким образом, *M(R*1*)* обозначает содержимое ячейки памяти, адрес которой определяется содержимым регистра *R*1.

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. 2.3.

Таблица 2.3.

Основные типы команд

Тип операции	Примеры
Арифметические и логические	Целочисленные арифметические и логические операции: сложение, вычитание, логическое сложение, логическое умножение и т. д.
Пересылки данных	Операции загрузки/записи
Управление потоком команд	Безусловные и условные переходы, вызовы процедур и возвраты из процедур
Системные операции	Системные вызовы, команды управления виртуальной памятью и т. д.

Операции с плавающей точкой	Операции сложения, вычитания, умножения и деления над вещественными числами
Десятичные операции	Десятичное сложение, умножение, преобразование форматов и т. д.
Операции над строками	Пересылки, сравнения и поиск строк

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.

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 (СК) 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).

2.3. Компьютеры со стековой архитектурой

При создании компьютера одновременно проектируют и систему команд 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 (1А), двухадресные (2А), трехадресные (3А), безадресные (БА), команды с большой длиной слова (VLIW – БДС) (рис. 2.1):

1А ~

КОП	А1

2А ~

КОП	А1	А2

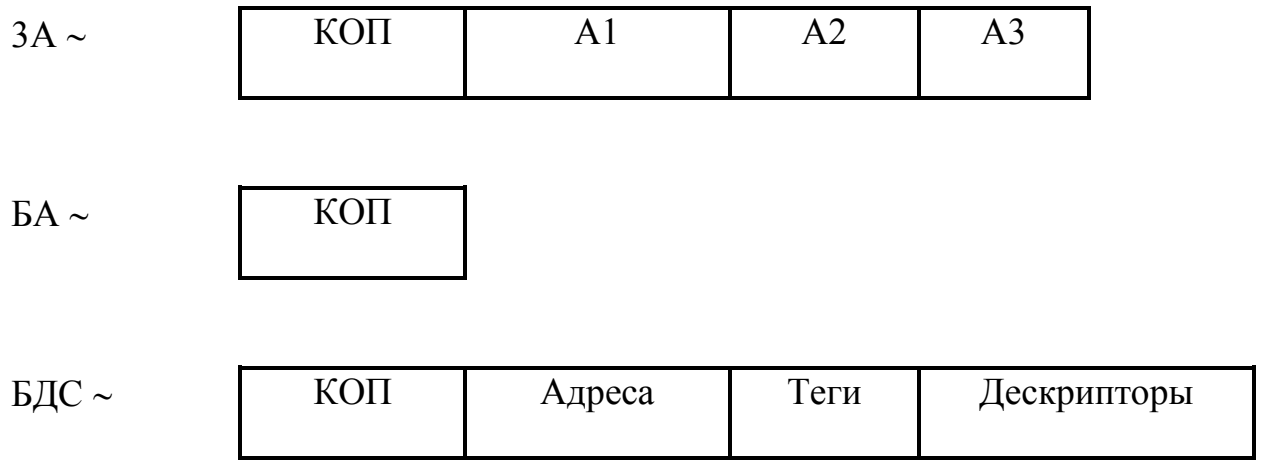

Рис. 2.1. Структуры команд

Причем операнд может указываться как адресом, так и непосредственно в структуре команды. В случае БА-команд операнды выбираются из стека, 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 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.

Вначале рассмотрим пример вычисления значения выражения

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

с помощью одноадресного компьютера. Последовательность необходимых команд приведена в табл. 2.4.

Таблица 2.4.

Последовательность команд для вычисления выражения

Номер команды	**Команда**	**Комментарии**
1	$c \to \sum$	*Выборка из памяти переменной* ***c*** *и загрузка в регистр* $\boldsymbol{\Sigma}$
2	$(\sum) + b$	*Выборка из памяти переменной* ***b****, выполнение операции сложения с содержимом регистра* $\boldsymbol{\Sigma}$ *и сохранение результата в регистре* $\boldsymbol{\Sigma}$
3	$(\sum) \to P_1$	*Сохранение результата сложения* в P_1 *– рабочей ячейке в памяти*
4	$a \to \sum$	*Выборка из памяти переменной* ***a*** *и загрузка в регистр* $\boldsymbol{\Sigma}$

5	$(\Sigma)\bullet a$	*Выполнение операции умножения с сохранением результата в регистре*
6	$(\Sigma)\to P_2$	*Сохранение результата сложения в* P_2 *– рабочей ячейке*
7	$b\to\Sigma$	*Выборка из памяти переменной* ***b*** *и загрузка в регистр* ***Σ***
8	$(\Sigma)\bullet b$	*Выполнение операции умножения с сохранением результата в регистре*
9	$(\Sigma)+(P_2)$	*и т.д.*
10	$(\Sigma):(P_1)$	$X=\dfrac{a^2+b^2}{b+c}\to\Sigma$

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

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 кеш памяти.

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. 2.2.

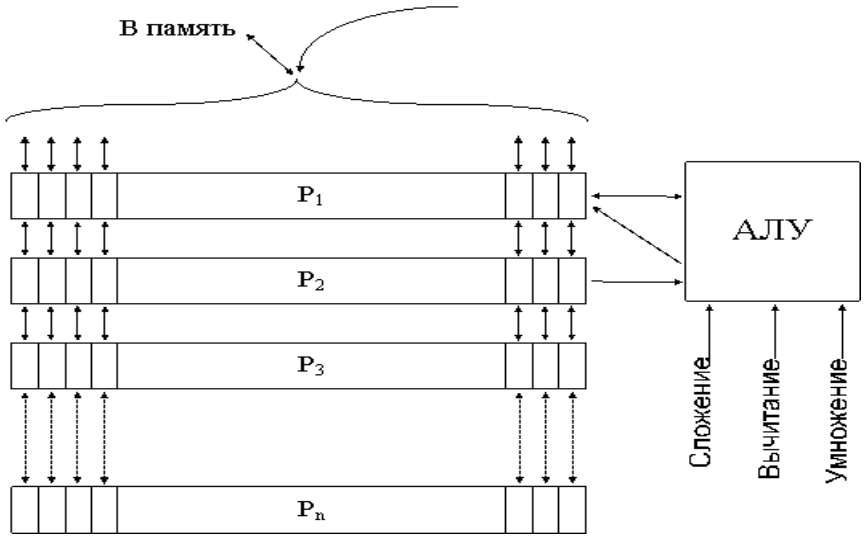

Рис. 2.2. Стековая организация процессора

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

Движение вниз: $(P_1) \to P_2$, $(P_2) \to P_3$, ..., а P_1 заполняется данными из главной памяти. Движение вверх: $(P_n) \to P_{n-1}$, $(P_{n-1}) \to P_{n-2}$, а P_n заполняется нулями.

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

Таким образом, арифметические операции 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 компьютерах со стековой памятью используются команды переменной длины.

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 вводятся специальные команды:

- ***дублирование*** ~ $(P_1) \to P_2$, $(P_2) \to P_3$, ... и т. д., а (P_1) остается при этом неизменным;

- ***реверсирование*** ~ $(P_1) \to P_2$, *а* $(P_2) \to P_1$, что удобно для выполнения некоторых операций.

Рассмотрим предыдущий пример вычисления выражения, но с использованием процессора со стековой организацией памяти (табл. 2.5):

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

Таблица 2.5.

Последовательность команд для процессора со стековой памятью

№	Команда	P_1	P_2	P_3	P_4
1	*Вызов b*	*b*			
2	*Дублирование*	*b*	*b*		
3	*Вызов c*	*c*	*b*	*b*	
4	*Сложение*	*b+c*	*b*		
5	*Реверсирование*	*b*	*b+c*		
6	*Дублирование*	*b*	*b*	*b+c*	
7	*Умножение*	b^2	*b+c*		
8	*Вызов a*	*a*	b^2	*b+c*	
9	*Дублирование*	*a*	*a*	b^2	*b+c*
10	*Умножение*	a^2	b^2	*b+c*	

11	*Сложение*	a^2+b^2	$b+c$		
12	*Деление*	$\frac{a^2+b^2}{b+c}$			
13	*Сохранение результата в памяти из регистра* **P_1**				

Как следует из табл. 2.5, понадобились лишь **три обращения к памяти** для вызова операндов (команды 1, 3, 8) и **одно обращение для записи** результата. Меньше обращений принципиально невозможно. Операнды и промежуточные результаты поступают для операций в АЛУ из стековой памяти; 9 команд из 13 являются безадресными.

Вся программа размещается в 4-**х 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 и 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.