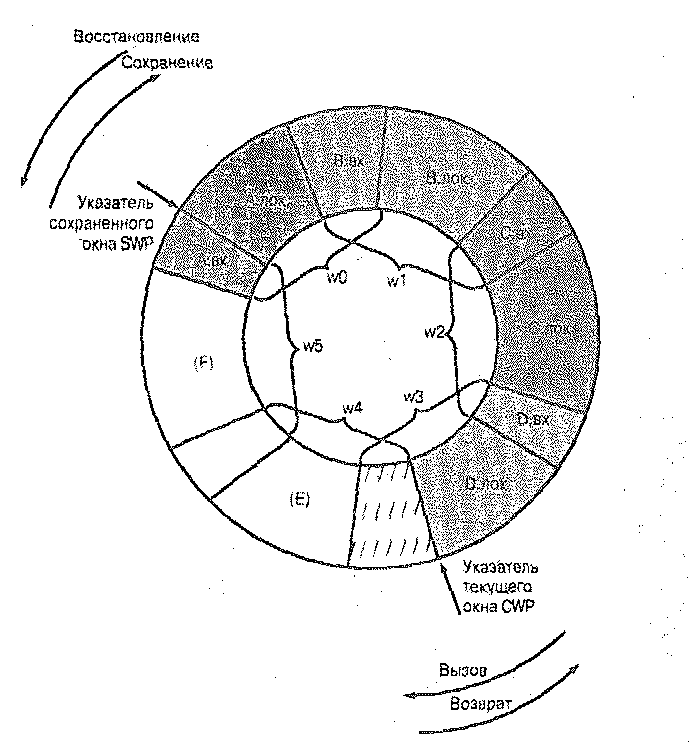
**Характеристики выполнения команд**

Иркутский государственный университет путей сообщения

Иркутск 2010



Анализ процесса выполнения программ, написанных на языках высокого уровня, создал предпосылки для разработки нового типа архитектуры процессоров — RISC-архитектуры. Ее особенностью является использование сокращенного набора машинных команд. Анализ показал» что доминирующими в программе являются операторы присваивания, а это означает, что основные усилия следует направить на оптимизацию операций передачи переменных. Кроме того, в программах встречается очень много условных выражений XF и операторов цикла LOOP, что требует разработки эффективного механизма управления, оптимизирующего конвейерную организацию выполнения ма-шинных команд. В то же время анализ форм адресации показал, что вполне возможно добиться высокой производительности работы процессора, размещая операнды в регистрах.

Результаты этих исследований и определили ключевые характеристики RISC-компьютеров; небольшое количество команд фиксированного формата в наборе, большое количество регистров или применение компиляторов, оптимизирующих использование регистров, упор на рациональную организацию работы конвейера операций.

Само по себе сокращение набора команд уже создает достаточно хорошие предпосылки для повышения эффективности работы конвейера, поскольку алго-ритм выполнения команд становится более регулярным и лучше предсказуемым. Кроме того, RISC-архитектура лучше подходит для применения задержанной технологии выполнения команд перехода и перекомпоновки других команд в программе, что также повышает эффективность работы конвейера.

С тех пор как в начале 1950-х годов были созданы первые вычислительные машины с хранимой программой, принципиально новых идей в области архитектуры и структурной организации компьютерных систем было не так много, и пересчитать их можно на пальцах.

• Концепция семейства машин. Эта концепция была внедрена впервые специалистами из IBM при проектировании семейства S/360 в 1964 году. За ними последовали разработчики из DEC со своим семейством PDP-8. Концепция семейства предусматривает определенное дистанцирование архитектуры компьютера от его структурной и схемной реализации. Потребителю предлагается функциональный ряд компьютеров, разных по производительности и стоимости, но имеющих одинаковую архитектуру.

• Микропрограммное управление. Этот принцип предложен М. Уилксом (M.V. Wiikes) в 1951 году и реализован в семействе IBM S/360 в 1964 году. Микропрограммирование облегчает разработку и упрощает структуру устройства управления процессора, а также хорошо сочетается с концепцией семейства компьютеров.

• Применение кэш-памяти. Реально впервые реализовано в модели IBM 360/85 в 1968 году. Включение уровня, кэш-памяти в иерархию памяти компьютера позволило существенно повысить его производительность.

• Конвейерная организация. Такая организация позволила на практике реализовать принцип совмещения операций при последовательном характере обработки команд программы. Примерами могут служить конвейер выполнения машинных команд и векторная обработка.

• Использование в единой системе множества "процессоров. Эта концепция имеет множество интерпретаций, отличающихся целевым назначением и структурной организацией.

К этому списку сейчас можно добавить и одну из наиболее интересных новейших идей, потенциально сулящую переворот в наших взглядах на архитектуру компьютеров — идею сокращенного набора команд. Переход на RISC-архитектуру означает кардинальное изменение многих существующих взглядов на принципы построения процессоров.

Хотя специалисты и не руководствуются единым общепринятым критерием принадлежности определенной системы к типу RISC-систем, большинство из них со-глашается с тем, что следует учитывать указанные ниже особенности организации:

• большое количество универсальных регистров в составе процессора или ориентация на применение компиляторов, оптимизирующих использование регистров;

• ограниченное количество относительно простых команд в наборе; .

• перенос центра усилий при проектировании на' оптимизацию конвейера операций.

В табл. 12.1 сравниваются параметры некоторых RISC-систем, CISC-систем1 и систем с суперскалярной архитектурой.

Эту главу мы начнем с небольшого обзора особенностей процесса выполнения программы и рассмотрим три только что упомянутых темы. Затем последует описание двух конкретных RISC-систем, по которым имеется широко доступная и подробная техническая документация.

1Аббревиатура CISC означает complex instruction set computer — компьютер с расширенным набором команд. — Прим, перев.

Таблица 12:1. Сравнительные характеристики процессоров с CISC-, RISC- и суперскалярной архитектурой

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | CISC-системы | | | RISC-системы | | Суперскалярные | | системы |
| Параметр | IBM 370/168 | VAX 11/780 | Intel 80486 | SPAEC | MIPS  R4000 | Power  PC | Ultra SPARC | MIPS  R10000 |
| Год разработки | 1973 | 1978 | 1989 | 1987 | 1991 | 1993 | 1996 | 1996 |
| Количество команд в наборе | 208 | 308 | 235 | 69 | 94 | 225 |  |  |
| Размер команды (байт) | 2-6 | 2-57 | 1-11 | 4 | 4 | 4 | 4 | 4 |
| Количество режимов адресации | 4 | 22 | 11 | 1 | 1 | 2 | 1 | 1 |
| Количество  универсальных регистров | 16 | 16 | 8 | 40-520 | 82' | 32 | 40-620 | 32 |
| Размер управляющей памяти (Кбит) | 420 | 480 | 246 | \_ | \_ | \_ | \_ | \_ |
| Размер кэш-памяти' (Кбайт) | 64 | 64 | 8 | 32 | 128 | 16-32 | 32 | 64 |

**Характеристики выполнения команд**

Наиболее заметно эволюция применения компьютеров во всех областях жизни общества отразилась на языках программирования. По мере того как доля стоимости аппаратной части в общей цене компьютерной системы снижается, доля программной части растет. Кроме того, хроническая нехватка специалистов по разработке программного обеспечения влечет за собой и рост стоимости программ в абсолютном выражении. Помимо увеличения стоимости и неудобства работы с некоторыми программами существует' еще и фактор ненадежности программного обеспечения — нередки случаи, когда в течение многих лет эксплуатации в программах обнаруживаются все новые и новые ошибки, причем это в равной мере относится и к системным, и к прикладным программам.

В ответ на запросы пользователей разных категорий разрабатываются все новые и новые языки программирования высокого уровня (ЯПВУ), которые позволяют в более компактной форме представлять алгоритмы, берут на себя заботы о деталях реализации вычислений и "часто поддерживают естественный для многих задач объектно-ориентированный характер обработки информации.

Но увы, повышение уровня "интеллектуальности" ЯПВУ порождает проблему, известную сейчас под именем "семантического разрыва" — существенного разрыва между операциями, описываемыми выражениями ЯПВУ, и операциями, поддерживаемыми на уровне машинного языка. Этот разрыв проявляется в снижении эффективности процесса выполнения программы, росте размеров машинного программного кода и усложнении программ-компиляторов. Специалисты, занятые разработкой архитектуры компьютеров, задались целью устранить разрыв. Среди причин его возникновения многие называют возросшее количество машинных команд в наборе, большое разнообразие режимов адресации, стремление аппаратно реализовать некоторые операторы ЯПВУ. Примером последней тенденции является машинная команда CASE в процессоре VAX. Использование расширенных наборов команд преследует такие цели:

• упрощение компиляторов;

• повышение эффективности выполнения программы, поскольку сложные последовательности операций могут быть реализованы на уровне микрокоманд;

• поддержка все более сложных и "интеллектуальных" ЯПВУ.

В последние годы проводилось множеств исследований с целью выявления закономерностей, процесса выполнения машинного кода, . который формируется после компиляции программ, написанных на ЯПВУ. Результаты этих исследований послужили для некоторых специалистов основой поиска нового подхода — добиваться повышения эффективности поддержки ЯПВУ не. за счет усложнения, а, наоборот, за счет упрощения архитектуры компьютера.

Чтобы понять ход рассуждений адептов нового подхода, обратимся к особенностям процесса выполнения машинной команды. Вас интересуют следующие аспекты этого процесса.

• Выполняемые операции — функции, которые возлагаются на процессор, и взаимодействие процессора с памятью.

• Используемые операнды — типы операндов и частота их использования в командах. Эта информация послужит для выбора множества режимов адресации.

• Последовательность выполнения. Эта информация определяет структуру управления процессором и конвейера операции.

В данном разделе мы рассмотрим результаты многочисленных исследований статистических характеристик программ .на языках высокого уровня. В процессе исследований анализировалось поведение программы во время ее. выполнения, что позволяет считать соответствующие, параметры динамическими, в отличие от статических, получаемых при анализе "чистого" текста программы. Статические параметры не несут информации, необходимой для оценки производительности работы компьютера, на котором выполняется программа, поскольку при определении этих параметров не учитывается, сколько раз выполняется тот или иной фрагмент программы в процессе решения задачи

**Операции**

Анализу частоты применения операторов определенных типов в программах на языках высокого уровня посвящено довольно много исследований, основные результаты которых приведены в табл. 4.9 (см. приложение 4А в конце главы 4). Как видно из этой таблицы, данные, полученные при исследовании программ разного назначения, написанных на • распространенных языках, достаточно хорошо согласуются. Доминирующее положение в 'программах занимают операторы присваивания, что указывает на важность простых средств пересылки данных в наборе машинных команд. Следующими по частоте появления в программах являются условные выражения вроде IF и LOOP. В машинных программах эти выражения реализуются командами сравнения и условного перехода. Отсюда следует, что при проектировании набора команд особое внимание должно быть уделено командам управления ходом выполнения программы. Результаты анализа структур программ служат исходной информацией разработчику набора машинных команд процессора, указывая, какие операции встречаются при выполнении программ чаще других, а следовательно, оптимальная реализация каких команд дает наиболее ощутимый эффект. Нужно отметить, что оценки, полученные при анализе текстов программ, не несут информации о том, сколько времени реально тратится в процессе выполнения программы на операции того или иного типа. Чтобы получить оценки динамических параметров, требуется анализировать процесс выполнения странсированной программы на машинном языке и выяснить, какие операторы в исходном тексте программы повлекли за собой включение в выполняемую машинную программу большей части команд.

Такой анализ был выполнен следующим образом. В исследовании Паттерсона (Patterson) [РАТТ82-а], о котором мы уже упоминали в главе 4, тестовые программы на языках С и Pascal были скомпилированы в машинные программы компьютеров VAX, PDP-11 и Motorola 68000. Затем в процессе выполнения программ оценивалось среднее количество машинных команд и обращений к памяти при реализации операторов - разных типов. Соответствующие данные представлены во второй ж третьей колонках табл. 12.2. Они были получены при регистрации частоты появления определенных групп команд в процессе выполнения программы и, следовательно, отражают динамику выполнения программы. Данные в четвертой и пятой колонках табл. 12.2 представляют взвешенные статистические оценки и были получены следующим образом: каждое значение из второй и третьей колонок умножалось на количество машинных команд, которое сформировал компилятор при трансляции оператора соответствующего типа в исходной программе. Затем результаты были нормализованы. Таким образом, каждый элемент в четвертой и пятой колонках представляет собой относительную частоту появления, оператора определенного типа в типовой программе, взятую с весом, пропорциональным количеству машинных команд, необходимых для его реализации в выполняемой программе. По этому же принципу сформированы и данные в шестой и седьмой колонках табл. 12, 2, но в качестве весовых коэффициентов взято количество обращений к памяти при реализации в машинной программе оператора каждого типа.

Данные в колонках с четвертой по седьмую можно рассматривать как косвенную оценку времени выполнения в машинной программе операторов того или иного типа из программы на языке высокого уровня. Из приведенных данных следует, что больше всего времени в типовой программе уходит на процедуры вызова подпрограмм и возврата из подпрограмм.

Обращаю внимание читателей на то, что данные в табл. 12.2 справедливы для наборов команд, типичных для компьютеров 80-х годов.. Вполне возможно, что выполненный по этой же методике анализ для компьютеров с другим набором машинных команд даст несколько отличающиеся результаты, но, тем не менее, эти данные .вполне репрезентативны для всех компьютеров с расширенным набором команд (компьютеров с CISC-архитектурой). Следовательно» основываясь на них, можно искать наиболее эффективные пути поддержки языков высокого уровня в наборе машинных команд.

Таблица 12.2. Взвешенная относительная динамическая частота появления в программах операторов языков высшего уровня

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Динамическая частота появления | | Взвешенная оценка количества машинных команд | | Взвешенная оценка количества обращений к памяти | |
|  | Pascal | С | Pascal | С | Pascal | С |
| Присваивание | 45 | 38 | 13 |  | 14 | 15 |
| (ASSIGN) |  |  |  |  |  |  |
| Возврат на | 5 | 3 | 42 | 32 | 83 | 26 |
| начало цикла |  |  |  |  |  |  |
| (LOOP) |  |  |  |  |  |  |
| Вызов | 15 | 12 | 31 | 33 | 44 | 45 |
| подпрограмм |  |  |  |  |  |  |
| (CALL) |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| Условный | 29 | 43 | 11 | 21 | 7 | 13 |
| переход (IF) |  |  |  |  |  |  |
| Безусловный |  |  |  |  |  |  |
| переход | - | 3 | - | - | - | - |
| (GOTO) |  |  |  |  |  |  |
| Другие | 6 | 1 | 3 | 1 | 2 | 1 |

Хотя статистические характеристики типов операндов, используемых в программах, не менее важны, чем характеристики . операторов, их исследованию уделялось значительно меньше внимания. Особый интерес представляют несколько аспектов применения операндов в программах.

В уже упоминавшейся работе Паттерсона [РАТТ82-а] рассматривалась динамическая частота появления в программах переменных разных классов (табл. 12.3). Статистические параметры для программ, написанных на языках С и Pascal, достаточно хорошо согласуются, и оказывается, что большинство ссылок в программе относится к скалярным переменным. Более того, свыше 80% этих скалярных переменных являются локальными. Ссылки на массивы или составные структурные переменные требуют предварительного обращения к соответствующему индексу или указателю, который, в свою очередь, чаще всего также является локальной переменной. Следовательно, в типичной программе превалируют ссылки на скалярные переменные, причем значительная часть ез. являются локальными.

В исследовании Паттерсона анализировались динамические статистические параметры программ на языках высокого уровня, не зависящие от того, какова архитектура компьютера, на котором эта программа будет выполняться. Как уже отмечалось в предыдущем разделе, при более глубоком статистическом анализе программ архитектуру компьютера следует обязательно учитывать. В работе [LUND77] рассматривались динамические характеристики программ компьютера PDP-10 и было показано, что в среднем на одну машинную команду приходится 0.5 ссылок на операнд, находящийся в памяти, и 1.4 ссылок на операнды в регистрах. Аналогичные результаты приводятся и в работе [HUCK83J, где анализировались программы компьютеров IBM S/370, PDP-11 и VAX, написанные на языках С, Pascal и FORTRAN. Конечно, эти характеристи-ки очень зависят от особенностей архитектуры компьютеров и качества компиляторов, но, тем не менее, они несут объективную информацию о соотношении разных способов обращения к переменным в программах.

Последние из упомянутых работ, с одной стороны, свидетельствуют о важности учета характеристик компьютера при анализе, а с другой — указывают, на какие способы обращения к операндам нужно обратить особое внимание конструкторам процессоров, поскольку они встречаются в программах чаще других. Из исследований Паттерсона следует, что в первую очередь нужно оптимизировать механизм обращения к локальным скалярным переменным.

Таблица 12.3. Динамическое распределение типов операндов в программах

|  |  |  |  |
| --- | --- | --- | --- |
|  | Pascal | C | В среднем |
| Целые константы | 16 | 23 | 20 |
| Скалярные переменные | 58 | 53 | 55 |
| Массивы/структуры | 26 | 24 | 25 |

**Вызовы подпрограмм**

Выше неоднократно подчеркивалось, что обращение к подпрограммам является одним из наиболее важных аспектов программирования на языках высокого уровня. Данные, , представленные в табл. 12.2, свидетельствуют о том, что больше всего времени при выполнении типичных программ тратится именно на операции вызова подпрограмм и возврата из подпрограмм. Следовательно, разработка методов оптимальной реализации таких операторов на уровне машинных команд принесет наибольший эффект. При выборе методов реализации нужно принимать во внимание два аспекта: количество параметров (аргументов), передаваемых при вызове подпрограммы, и переменных, с которыми работает подпрограмма; глубина вложенности вызовов подпрограмм.

В своей работе [TANE78] Таненбаум (Tanenbaum) показал, что при вызове 98% подпрограмм передается менее 6 параметров и что в 92% вызванных подпрограмм' используется менее 6 скалярных локальных переменных. Аналогичные результаты получила и группа исследователей из Беркли [КАТЕ83] (табл. 12.4). Они свидетельствуют, что для активизации подпрограмм нужно передавать весьма ограниченное количество слов данных. Работы, на которые мы ссылались выше, также указывают на то, что большая часть операций внутри подпрограмм выполняется с локальными скалярными переменными, причем количество таких переменных в типичном случае весьма невелико.

Группа исследователей RISC-архитектуры из Беркли также проанализировала типичную последовательность вызовов подпрограмм и возврата из них в главной программе, написанной на языке высокого уровня. Было обнаружено, что в программах довольно редко встречается непрерывная последовательность вызовов вложенных подпрограмм и, соответственно, непрерывная последовательность операторов возврата из подпрограмм. Значительно чаще программа имеет дело со сравнительно узким окном вложенности. Ранее, в главе 4, мы уже иллюстрировали это свойство типичных программ диаграммой, представленной на рис. 4.29. Результаты упомянутых исследований еще раз подтверждают уже неоднократно описанное свойство локализации ссылок, присущее подавляющему большинству компьютерных программ.

Таблица 12.4. Аргументы ж локальные переменные подпрограмм

|  |  |  |
| --- | --- | --- |
| Характеристика выполняемой подпрограммы | Компиляторы, интерпретаторы | Небольшие программы,  не включающие численных расчетов |
| Более 3 аргументов | 0-7% | 0-5% |
| Более 6 аргументов | 0-3% | 0% |
| Более 8 слов аргументов и локальных скалярных переменных | 1-20% | 0-6% |
| Более 12 слов аргументов и локальных скалярных переменных | 1-6% | 0-3% |

**Реализация**

Описанные выше результаты привели большинство исследователей к заключению, что разрабатывать набор машинных команд, близкий к операторам языков высокого уровня, — это отнюдь не самая эффективная стратегия. Многие обращают внимание на целесообразность поиска путей оптимальной реализации групп машинных команд, которые соответствуют тем операторам языков высокого уровня, выполнение которых в типичной программе занимает больше всего времени. В результате обобщения проведенных исследований были сформулированы три характерных черты RISC-архитектуры.

Во-первых, использование большого количества регистров в составе процессора или применение компиляторов, оптимизирующих работу с регистрами в машинной программе. Это должно привести к повышению эффективности механизма обращения к операндам. Упомянутые выше исследования также свидетельствуют, что значительная часть ссылок на операнды при этом приходится на операторы пересылки данных (операторы присваивания в языках высокого уровня). Поскольку для программ характерна локализация ссылок и доминирование локальных скалярных переменных, то эффективным путем повышения производительности программ должно стать сокращение количества обращений к переменным, хранящимся в памяти, и более интенсивное использование переменных, хранящихся в регистрах процессора. Поскольку подавляющее большинство ссылок локализовано, для этого вполне реально включить в состав процессора расширенный набор регистров.

Во-вторых, серьезное внимание должно быть уделено организации конвейера выполнения машинных команд. Поскольку в программах значительное место занимают- команды, условного перехода и вызова подпрограмм, простой механизм конвейерного выполнения, будет неэффективным в виду того, что от довольно большой части заранее выполненных команд потом придется отказаться.

И в-третьих, из приведенного анализа следует, что в процессорах можно использовать сокращенный набор команд. Этот вывод не кажется очевидным, но мы постараемся убедить вас в его справедливости по мере дальнейшего изложения.