Тесты кафедры анатомии человека МГМСУ им. А.И. Евдокимова

Содержание

Введение

Вычислительная система - это совокупность аппаратных (hardware) и представленных в определенном формате математических средств (software), реализующих процесс решения задач. Другими словами, это вычислительная среда, на которую отображаются (в которой моделируются) задачи реального мира.

Для моделирования реальных задач можно использовать два подхода, определяющих архитектуру вычислительной среды.

Первый подход, так называемый фундаментальный, базируется на классической математике, как средстве описания «идейной» сути объекта исследования. Это аналитический подход, предполагающий декомпозицию моделируемого объекта на элементарные явления, описываемые простыми уравнениями, имеющими ясную физическую интерпретацию, и дальнейшее объединение этих уравнений в единую математическую модель в соответствии с законами, присущими данной предметной области.

Построение и исследование таких моделей может выполняться в среде цифровой микропроцессорной системы, основанной на принципах архитектуры фон Неймана. Последовательный цифровой принцип вычислений, базирующийся на формальном аппарате алгебры логики, отличается универсальностью, обеспечивает необходимую точность вычислений и как пути повышения их эффективности использует:

-увеличение тактовой частоты процессоров;

-конвейеризацию выполнения инструкций;

-распараллеливание вычислительных структур на уровне команд и на уровне алгоритмов, а также ряд других архитектурных усовершенствований, способствующих повышению эффективности вычислительного процесса.

Второй подход, эмпирический, исключает, как базовые, фазу анализа и аналитическое представление модели. Это синтетический подход, при котором

объект (задача) моделируется сразу целиком (как «черный ящик») в реальных условиях без выделения идеальных элементарных явлений и проникновения в математическую суть исследуемой задачи. Вычислительной средой, реализующей данный подход, могут быть искусственные нейронные сети (ИНС), лежащие в основе «существенно параллельных» вычислительных систем - нейрокомпьютеров. В идеальной реализации это аналоговый подход с присущей ему высокой скоростью вычислений, но вместе с тем предоставляющий возможность достижения приемлемой точности путем:
-повышения мощности нейронной сети (увеличения числа нейронов, количества слоев);
-совершенствования ее архитектуры;
-применения эффективных процедур обучения;
-сочетания с элементами цифровой вычислительной техники.
Архитектура вычислительной системы
С середины 60-х годов очень сильно изменился подход к созданию вычислительных машин. Вместо разработки аппаратуры и средств математического обеспечения стала проектироваться система, состоящая из синтеза аппаратных (hardware) и программных (software) средств. При этом на главный план выдвинулась концепция взаимодействия. Так возникло новое понятие -- архитектура вычислительной системы (ВС).
Под архитектурой ВС принято понимать совокупность общих принципов организации аппаратно-программных средств и их основных характеристик, определяющая функциональные возможности вычислительной машины при решении соответствующих типов задач.
Архитектура ВС охватывает значительный круг проблем, связанных с созданием комплекса аппаратных и программных средств и учитывающих большое количество определяющих факторов. Среди этих факторов основными являются: стоимость, сфера применения, функциональные возможности, удобство в эксплуатации, а одним из основных компонентов архитектуры считаются аппаратные средства.
Архитектуру вычислительного средства необходимо отличать от структуры ВС. Структура вычислительного средства определяет его текущий состав на определенном уровне детализации и описывает связи внутри средства. Архитектура же определяет основные правила взаимодействия составных элементов вычислительного средства, описание которых выполняется в той мере, в какой необходимо для формирования правил взаимодействия. Она устанавливает не все связи, а только наиболее необходимые, которые должны быть известны для более грамотного использования применяемого средства.
Так, пользователю ЭВМ не важно, на каких элементах выполнены электронные схемы, схемно или программно исполняются команды и тому подобное. Архитектура ВС действительно отражает круг проблем, которые относятся к общему проектированию и построению вычислительных машин и их ПО.
Архитектура ВС включает в себя как структуру, отражающую состав ПК, так и программно - математическое обеспечение. Структура ВС - совокупность элементов и

связей между ними. Основным принципом построения всех современных ВС является программное управление.
Основы учения об архитектуре вычислительных машин были заложены Джоном фон Нейманом. Совокупность этих принципов породила классическую (фон-неймановскую) архитектуру ВС.
Фон Нейман не только выдвинул основополагающие принципы логического устройства ВС, но и предложил ее структуру, представленную на рисунке 1.
Рисунок 1.
Положения фон Неймана:
Компьютер состоит из нескольких основных устройств (арифметико-логическое устройство, управляющее устройство, память, внешняя память, устройства ввода и вывода)
Арифметико-логическое устройство - выполняет логические и арифметические действия, необходимые для переработки информации, хранящейся в памяти.
Управляющее устройство - обеспечивает управление и контроль всех устройств компьютера (управляющие сигналы указаны пунктирными стрелками).
Данные, которые хранятся в запоминающем устройстве, представлены в двоичной форме.
Программа, которая задает работу компьютера, и данные хранятся в одном и том же запоминающем устройстве.
Для ввода и вывода информации используются устройства ввода и вывода.
Современную архитектуру компьютера определяют следующие принципы:
Принцип программного управления. Обеспечивает автоматизацию процесса вычислений на ЭВМ. Согласно этому принципу, для решения каждой задачи составляется программа, которая определяет последовательность действий компьютера. Эффективность программного управления будет выше при решении задачи этой же программой много раз (хотя и с разными начальными данными).
Принцип программы, сохраняемой в памяти. Согласно этому принципу, команды программы подаются, как и данные, в виде чисел и обрабатываются так же, как и числа, а сама программа перед выполнением загружается в оперативную память, что ускоряет процесс ее выполнения.
Принцип произвольного доступа к памяти. В соответствии с этим принципом, элементы программ и данных могут записываться в произвольное место оперативной памяти, что позволяет обратиться по любому заданному адресу (к конкретному участку памяти) без просмотра предыдущих.
На основании этих принципов можно утверждать, что современный компьютер - техническое устройство, которое после ввода в память начальных данных в виде цифровых кодов и программы их обработки, выраженной тоже цифровыми кодами, способно автоматически осуществить вычислительный процесс, заданный программой, и выдать готовые результаты решения задачи в форме, пригодной для восприятия человеком.
Реальная структура компьютера значительно сложнее, чем рассмотренная выше (ее можно назвать логической структурой). В современных компьютерах, в частности

персональных, все чаще происходит отход от традиционной архитектуры фон Неймана, обусловленный стремлением разработчиков и пользователей к повышению качества и производительности компьютеров. Качество ВС характеризуется многими показателями. Это и набор команд, которые компьютер способный понимать, и скорость работы (быстродействие) центрального процессора, количество периферийных устройств ввода-вывода, присоединяемых к компьютеру одновременно и т.д. Главным показателем является быстродействие - количество операций, какую процессор способен выполнить за единицу времени. На практике пользователя больше интересует производительность компьютера - показатель его эффективного быстродействия, то есть способности не просто быстро функционировать, а быстро решать конкретные поставленные задачи.
Как результат, все эти и прочие факторы способствуют принципиальному и конструктивному усовершенствованию элементной базы компьютеров, то есть созданию новых, более быстрых, надежных и удобных в работе процессоров, запоминающих устройств, устройств ввода-вывода и т.д. Тем не менее, следует учитывать, что скорость работы элементов невозможно увеличивать беспредельно (существуют современные технологические ограничения и ограничения, обусловленные физическими законами). Поэтому разработчики компьютерной техники ищут решения этой проблемы усовершенствованием архитектуры ВС.
Так, появились компьютеры с многопроцессорной архитектурой, в которой несколько процессоров работают одновременно, а это означает, что производительность такого компьютера равняется сумме производительностей процессоров. В мощных компьютерах, предназначенных для сложных инженерных расчетов и систем автоматизированного проектирования (САПР), часто устанавливают два или четыре процессора. В сверхмощных ЭВМ (такие машины могут, например, моделировать ядерные реакции в режиме реального времени, прогнозировать погоду в глобальном масштабе) количество процессоров достигает нескольких десятков.
Скорость работы компьютера существенным образом зависит от быстродействия оперативной памяти. Поэтому, постоянно ведутся поиски элементов для оперативной памяти, затрачивающих меньше времени на операции чтения-записи. Но вместе с быстродействием возрастает стоимость элементов памяти, поэтому наращивание быстродействующей оперативной памяти нужной емкости не всегда приемлемо экономически.
Проблема решается построением многоуровневой памяти. Оперативная память состоит из двух-трех частей: основная часть большей емкости строится на относительно медленных (более дешевых) элементах, а дополнительная (так называемая кэш-память) состоит из быстродействующих элементов. Данные, к которым чаще всего обращается процессор находятся в кэш-памяти, а больший объем оперативной информации хранится в основной памяти.
Раньше работой устройств ввода-вывода руководил центральный процессор, что занимало немало времени. Архитектура современных компьютеров предусматривает наличие каналов прямого доступа к оперативной памяти для обмена данными с

устройствами ввода-вывода без участия центрального процессора, а также передачу большинства функций управления периферийными устройствами специализированным процессорам, разгружающим центральный процессор и повышающим его производительность.
Рассмотрим несколько способов ускорения традиционных архитектур вычислительных систем.
Способы ускорения традиционных архитектур вычислительных систем
Операционные конвейеры.
Разработчики архитектуры компьютеров издавна прибегали к методам проектирования, известным под общим названием "совмещение операций", при котором аппаратура компьютера в любой момент времени выполняет одновременно более одной базовой операции. Этот общий метод включает два понятия: параллелизм и конвейеризацию. Хотя у них много общего и их зачастую трудно различать на практике, эти термины отражают два совершенно различных подхода. При параллелизме совмещение операций достигается путем воспроизведения в нескольких копиях аппаратной структуры. Высокая производительность достигается за счет одновременной работы всех элементов структур, осуществляющих решение различных частей задачи. Конвейеризация (или конвейерная обработка) в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры. Так обработку любой машинной команды можно разделить на несколько этапов (несколько ступеней), организовав передачу данных от одного этапа к следующему. При этом конвейерную обработку можно использовать для совмещения этапов выполнения разных команд. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд. Конвейерная обработка такого рода широко применяется во всех современных быстродействующих процессорах. Конвейеризация увеличивает пропускную способность процессора (количество команд, завершающихся в единицу времени), но она не сокращает время выполнения отдельной команды. В действительности, она даже несколько увеличивает время выполнения каждой команды из-за накладных расходов, связанных с управлением регистровыми станциями. Однако увеличение пропускной способности означает, что программа будет выполняться быстрее по сравнению с простой неконвейерной схемой. Тот факт, что время выполнения каждой команды в конвейере не уменьшается, накладывает некоторые ограничения на практическую длину конвейера. Кроме ограничений, связанных с задержкой конвейера, имеются также ограничения, возникающие в результате несбалансированности задержки на каждой его ступени и из-за накладных расходов на конвейеризацию. Частота синхронизации не может быть выше, а, следовательно, такт синхронизации не может быть меньше, чем время, необходимое для работы наиболее медленной ступени конвейера. Накладные расходы на организацию конвейера возникают из-за задержки сигналов в конвейерных регистрах (защелках) и из-за перекосов сигналов синхронизации. Конвейерные регистры к длительности

такта добавляют время установки и задержку распространения сигналов. В предельном случае длительность такта можно уменьшить до суммы накладных расходов и перекоса сигналов синхронизации, однако при этом в такте не останется времени для выполнения полезной работы по преобразованию информации. При реализации конвейерной обработки возникают ситуации, которые препятствуют выполнению очередной команды из потока команд в предназначенном для нее такте. Такие ситуации называются конфликтами.
Конфликты снижают реальную производительность конвейера Существуют три класса конфликтов:
1. Структурные конфликты, которые возникают из-за конфликтов по ресурсам, когда аппаратные средства не могут поддерживать все возможные комбинации команд в режиме одновременного выполнения с совмещением.
2. Конфликты по данным, возникающие в случае, когда выполнение одной команды зависит от результата выполнения предыдущей команды. 3. Конфликты по управлению, которые возникают при конвейеризации команд переходов и других команд, которые изменяют значение счетчика команд.
Конфликты в конвейере приводят к необходимости приостановки выполнения команд (pipeline stall). Обычно в простейших конвейерах, если приостанавливается какая-либо команда, то все следующие за ней команды также приостанавливаются. Команды, предшествующие приостановленной, могут продолжать выполняться, но во время приостановки не выбирается ни одна новая команда.
Расслоение памяти.
Другой способ повышения пропускной способности оперативной памяти связан с построением памяти, состоящей на физическом уровне из нескольких модулей (банков) с автономными схемами адресации, записи и чтения. При этом на логическом уровне управления памятью организуются последовательные обращения к различным физическим модулям. Обращения к различным модулям могут перекрываться, и т.о. образуется своеобразный конвейер. Эта процедура носит название расслоения памяти. Целью данного метода является увеличение скорости доступа к памяти посредством совмещения фаз обращений ко многим модулям памяти. Существуют несколько вариантов организации расслоения. Наиболее часто используемый способ - расслоение обращений за счет расслоения адресов. Этот способ основывается на свойстве локальности программ и данных, предполагающем, что адрес следующей команды на 1 больше адреса предыдущей (иными словами, линейность программы нарушается только командами перехода). Аналогичная последовательность адресов генерируется процессором при чтении слов данных. Т.о., типичный случай распределения адресов - последовательность вида a, a+1, a+2, a+3 и т.д. (для слов данных - увеличение на 1 - условно, на самом деле 1 - число байт в машинном слове). Из этого следует, что расслоение обращений возможно, если ячейки с адресами a, a+1, a+2, a+3 и т.д будут размещаться в блоках 0,1,2… Такое распределение ячеек по модулям (банкам) обеспечивается за счет использования адресов вида (см. рис. 2).
Здесь B - k-разрядный адрес модуля (младшая часть m-разрядного адреса), C - n-

разрядный адрес ячейки в модуле В (старшая часть адреса).
Рис. 2. Формат адреса при организации расслоении обращений к памяти за счет расслоения адресов.
Принцип расслоения адресов иллюстрирован на рис 3. (а).
Рис 3 .Организация адресного пространства при расслоении памяти (а), временная диаграмма работы модулей (б)
Все команды и данные размещены в адресном пространстве последовательно. Однако ячейки памяти, имеющие смежные адреса, находятся в различных модулях памяти. Если ОП состоит из 4-х модулей, то номер модуля кодируется двумя младшими разрядами адреса. При этом полные m - разрядные адреса 0,4,8,…. Относятся к блоку 0, адреса 1,5,9,13 - к блоку 1, адреса 2,6,10 - к блоку 2 и адреса 3,7,11,15 - к блоку 3. Т.о., последовательность обращений к адресам 0,1,2,3,4,5,6….будет расслоена между 4 модулями : 0,1,2,3,0,1,2…(см рис 4)
0, 1, 2, 3, 4, 5, 60, 1, 2, 3, 0, 1, 2…..
Рис. 4 Расслоение последовательности обращений к адресам между модулями памяти
Поскольку каждый модуль памяти имеет собственные схемы управления выборкой, можно обращение к следующему модулю производить, не дожидаясь ответа от предыдущего.
На временной диаграмме видно, что время доступа к каждому модулю равно: ф =4Т, где Т=t i+1-ti - длительность такта. В каждом такте следуют обращения к модулям памяти в моменты времени t1,t2,t3,…. При наличии 4-х модулей темп выдачи квантов информации из памяти в процессор будет соответствовать 1 такту Т, при этом скорость выдачи информации из каждого модуля в 4 раза ниже, т.е. составит 4Т. Задержка в выдаче кванта информации относительно момента обращения к модулю также составит 4 такта, однако задержка в выдаче каждого последующего кванта относительно момента выдачи предыдущего составит Т.
При реализации расслоения по адресам число модулей памяти может быть произвольным и необязательно кратным 2. В некоторых компьютерах предусмотрено произвольное отключение модулей памяти, что позволяет исключить из конфигурации неисправные модули.
В современных высокопроизводительных компьютерах число модулей составляет 4-16, но иногда превышает 64.
Для повышения производительности мультипроцессорных систем, работающих в многозадачных режимах, применяют другие методы расслоения, при которых разные процессоры обращаются к различным модулям памяти.
Необходимо помнить, что процессоры ввода-вывода также занимают циклы памяти и вследствие этого могут сильно влиять на производительность системы. Для уменьшения этого влияния обращения ЦП и процессоров ввода-вывода организуют к разным модулям памяти.
Обобщением идеи расслоения памяти является возможность реализации нескольких независимых обращений, когда несколько контролеров памяти позволяют модулям памяти (или группам расслоенных модулей памяти) работать независимо.

Прямое уменьшение числа конфликтов чередующихся при обращении к памяти может быть достигнуто путем размещения программ данных в разных модулях. Поскольку обращения к командам и элементам данных чередуются, то разделение памяти на память команд и память данных повышает быстродействие машины подобно рассмотренному выше механизму расслоения. Разделение памяти на память команд и память данных широко используется в системах управления или обработки сигналов. В подобного рода системах в качестве данных используются ПЗУ, цикл которых меньше цикла устройств, допускающих запись. Такое решение делает разделение данных и команд весьма эффективным.

Выбор той или иной схемы расслоения для компьютерной и др. вычислительной системы определяется целями (достижение высокой производительности при решении множества задач или высокого быстродействия при решении одной задачи), а также архитектурными и структурными особенностями системы и элементной базой (соотношением длительности циклов памяти и узлов обработки).

Кэш-память.

В функциональном отношении кэш-память рассматривается как буферное ЗУ, размещённое между основной (оперативной) памятью и процессором. Основное назначение кэш-памяти - кратковременное хранение и выдача активной информации процессору, что сокращает число обращений к основной памяти, скорость работы которой меньше, чем кэш-памяти.

За единицу информации при обмене между основной памятью и кэш-памятью принята строка, причём под строкой понимается набор слов, выбираемый из оперативной памяти при одном к ней обращении. Хранимая в оперативной памяти информация представляется, таким образом, совокупностью строк с последовательными адресами. В любой момент времени строки в кэш-памяти представляют собой копии строк из некоторого их набора в ОП, однако расположены они необязательно в такой же последовательности, как в ОП.

Построение кэш-памяти может осуществляться по различным принципам, которые будут рассмотрены ниже.

Типовая структура кэш-памяти

Рассмотрим типовую структуру кэш-памяти (рис. 5), включающую основные блоки, которые обеспечивают её взаимодействие с ОП и центральным процессором.

Рис. 5. Типовая структура кэш-памяти

Строки, составленные из информационных слов, и связанные с ними адресные теги хранятся в накопителе, который является основой кэш-памяти. Адрес требуемого слова, поступающий от центрального процессора (ЦП), вводится в блок обработки адресов, в котором реализуются принятые в данной кэш-памяти принципы использования адресов при организации их сравнения с адресными тегами. Само сравнение производится в блоке сравнения адресов (БСА), который конструктивно совмещается с накопителем, если кэш-память строится по схеме ассоциативной памяти. Назначение БСА состоит в выявлении попадания или промаха при обработке запросов от центрального процессора. Если имеет место кэш-попадание (т.е. искомое слово хранится в кэш-памяти, о чём свидетельствует совпадение кодов адреса,

поступающего от центрального процессора, и одного из адресов некоторого адресного тега), то соответствующая строка из кэш-памяти переписывается в регистр строк. С помощью селектора-демультиплексора из неё выделяется искомое слово, которое и направляется в центральный процессор. В случае промаха с помощью блока формирования запросов осуществляется инициализация выборки из ОП необходимой строки. Адресация ОП при этом производится в соответствии с информацией, поступившей от центрального процессора. Выбираемая из памяти строка вместе со своим адресным тегом помещается в накопитель и регистр строк, а затем искомое слово передается в центральный процессор.
Для высвобождения места в кэш-памяти с целью записи выбираемой из ОП строки одна из строк удаляется. Определение удаляемой строки производится посредством блока замены строк, в котором хранится информация, необходимая для реализации принятой стратегии обновления находящихся в накопителе строк.
Способы размещения данных в кэш-памяти.
Существует четыре способа размещения данных в кэш-памяти:
-прямое распределение,
-полностью ассоциативное,
-частично ассоциативное,
-распределение секторов.
Рассмотрим подробно каждый способ размещения и механизмы преобразования адресов. Предположим, что кэш содержит 128 строк, размер строки 16 слов, а основная память может содержать 16384 строки. Для адресации основной памяти используется 18 бит. Из них 14 старших показывают адрес строки, а младшие 4 - адрес слова внутри этой строки. Строки КЭШ-памяти указываются 7-разрядными адресами.
Прямое распределение.
При прямом распределении место хранения строк в кэш-памяти однозначно определяется по адресу строки. Структура кэш-памяти с прямым распределением показана на рис. 6
Рис. 6. Структура кэш-памяти с прямым распределением
Адрес основной памяти состоит из 14-ти разрядного адреса строки и 4-х разрядного адреса слова внутри этой строки. Адрес строки подразделяется на старшие 7 бит (тег) и младшие 7 бит (индекс). Для того чтобы поместить в кэш-память строку из основной памяти с адресом АВС, выбирается область внутри кэш-памяти с адресом В, который равен 7 младшим битам адреса строки АВ. Преобразование из АВС в В сводится только к выборке младших 7 бит адреса строки АВ. По адресу В в кэш-памяти может быть помещена любая из 128 строк основной памяти, имеющих адрес, 7 младших бит которого равны адресу В. Для того, чтобы определить, какая именно строка хранится в памяти данных в настоящий момент времени, используется запоминающее устройство емкостью 7*128 слов, в котором помещается по соответствующему адресу в качестве тега 7 старших бит адреса строки, хранящейся в данное время по адресу В кэш-памяти. Это запоминающее устройство называется теговой памятью. Память, в которой хранятся строки, называется памятью данных.

Тег из теговой памяти считывается по адресу B, который образует 7 младших бит адреса строки AB. Параллельно считыванию тега осуществляется доступ к памяти данных с помощью 11 младших бит (BC) адреса основной памяти ABC. Если тег и старшие 7 бит адреса основной памяти совпадают, значит что данная строка существует в памяти данных (строка-V), то есть осуществляется кэш-попадание. Если же происходит кэш-промах, то есть тег отличается от старших 7 бит, то из основной памяти считывается соответствующая строка, а из кэш-памяти удаляется строка-V, определяемая 7 младшими разрядами адреса строки, а на ее место помещается строка, считанная из основной памяти. Осуществляется также обновление соответствующего тега в теговой памяти. Способ прямого распределения реализуется довольно просто, однако из-за того, что место хранения строки в кэш-памяти однозначно определяется по адресу строки, вероятность сосредоточения областей хранения строк в некоторой части кэш-памяти высока, то есть замены строк будут происходить довольно часто. В этой ситуации эффективность кэш-памяти заметно снижается.

Полностью ассоциативное распределение

При таком способе размещения данных каждая строка основной памяти может быть размещена на месте любой строки кэш-памяти. Структура кэш-памяти с полностью ассоциативным распределением выглядит как показано на рис 7.

Рис. 7. Структура кэш-памяти с полностью ассоциативным распределением

При полностью ассоциативном распределении механизм преобразования адресов должен давать ответ на вопрос, существует ли копия строки с произвольным адресом в кэш-памяти, и, если существует, то по какому адресу. Для этого необходимо, чтобы теговая память являлась ассоциативной памятью. Входной информацией для ассоциативной памяти тегов является тег A (14-ти разрядный адрес строки), а выходной информацией - адрес строки внутри кэш-памяти (C). Каждое слово теговой памяти состоит из 14-разрядного тега и 7-разрядного адреса C строки внутри кэш-памяти. Ключом для поиска адреса строки внутри кэш-памяти является тег A (старшие 14 разрядов адреса основной памяти). При совпадении ключа A с одним из тегов T теговой памяти (случай попадания) происходит выборка соответствующих данному тегу адреса C и обращение к памяти данных. Входной информацией для памяти данных является 11-ти разрядное слово BC (7 бит адреса строки B + 4 бита адреса слова в данной строке C). В случае несовпадения ключа ни с одним из тегов теговой памяти (случай промаха) формируется запрос к основной памяти на выборку строки с соответствующим адресом и считывание этой строки. По этому способу при замене строк кандидатом на удаление могут быть все строки в кэш-памяти.

Частично ассоциативное распределение.

При данном способе размещения, несколько соседних строк (фиксированное число, не менее двух) из 128 строк кэш-памяти образуют структуру называемую группой. Структура кэш-памяти, основанная на использовании частично ассоциативного распределения, показана на рис. 8. В данном случае в одну группу E входят 4 строки A, B, C, D.

Рис. 8. Структура кэш-памяти, основанная на использовании частично

ассоциативного распределения.
Адрес строки HE основной памяти (14 бит) разделяется на две части: H-тег (старшие 9 бит) и E - адрес группы (младшие 5 бит). Адрес строки внутри кэш-памяти, состоящий из 7 бит, разделяется на адрес группы E (5 бит) и адрес строки внутри группы (2 бит: 00,01,10,11).
Для помещения в кэш-память строки, хранимой в ОП по адресу HEF, необходимо выбрать группу с адресом E. При этом не имеет значения, какая из четырех строк в группе может быть выбрана. Для выбора группы используется метод прямого распределения, а для выбора строки в группе используется метод полностью ассоциативного распределения.
Когда центральный процессор запрашивает доступ по адресу HEF, то осуществляется обращение к массиву тегов по адресу E, выбирается группа из четырёх тегов (a, b, c, d), каждый из которых сравнивается со старшими 9 битами (H) адреса строки. На выходе четырех схем сравнения формируется унитарный код совпадения (H=A - код: 1000, H=B - код: 0100, H=C - код: 0010, H=D - код: 0001), который на шифраторе преобразуется в двухразрядный позиционный код, служащий адресом для выбора банка дан-ных (00,01,10,11) - адрес строки внутри группы.
Одновременно осуществляется обращение к массиву данных (банкам V1, V2, V3, V4,) по адресу EF (9 бит) и считывание из банка V2 требуемой строки или слова.
При пересылке новой строки в кэш-память удаляемая из нее строка выбирается из четырех строк соответствующего набора (группы).
Распределение секторов.
По этому способу основная память разбивается на секторы, состоящие из фиксированного числа строк, кэш-память также разбивается на секторы, состоящие из такого же числа строк. Допустим, в секторе 16 строк, а в строке - 16 слов.
Структура кэш-памяти с распределением секторов представлена на рис. 9
В адресе основной памяти 10 старших бит задают адрес сектора A, следующие 4 бита - адрес строки B в секторе и младшие 4 бита - адрес слова C в строке.
При данной организации кэш-памяти, распределение секторов в кэш-памяти и основной памяти осуществлено полностью ассоциативно, то есть, каждый сектор A основной памяти может соответствовать любому сектору D в кэш-памяти. К каждой строке V, хранящейся в кэш-памяти, добавляется один бит достоверности (действительности); он показывает, совпадает или нет содержимое этой строки с содержи-мым строки в основной памяти, которая в данный момент анализируется на соответствие строки кэш-памяти. Если слова, запрашиваемого центральным процессором при доступе, не существует в кэш-памяти (бит достоверности, выбранный по адресу BD равен 0), то сначала центральный процессор проверяет, был ли сектор A, содержащий это слово, помещен ранее в кэш-память. Если он отсутствует, то один из секторов кэш-памяти заменяется на этот сектор.
Рис. 9. Структура кэш-памяти с распределением секторов.
Если все сектора кэш-памяти используются, то выбирается один какой-нибудь сектор, и при необходимости только некоторые строки этого сектора возвращаются в основную память, а этот сектор можно использовать дальше.

Когда осуществляется доступ к сектору А в кэш-памяти и строка В, содержащая нужное слово С, пересылается из основной памяти, то бит достоверности устанавливается до пересылки строки. Все биты достоверности других строк этого сектора сбрасываются. Если сектор А, содержащий слово В доступ к которому запрашивается, уже находится в кэш-памяти, то, в том случае когда бит достоверности строки, содержащей это слово, равен 0, этот бит устанавливается и строка пересылается из основной памяти в данную область кэш-памяти. В том случае, когда бит достоверности уже равен 1, нужное слово можно считать из кэш-памяти.

Методы обновления строк в основной памяти

В таблице 1 приведены условия сохранения и обновления информации в ячейках кэш-памяти и основной памяти.

Если процессору требуется информация из некоторой ячейки основной памяти, а копия этой ячейки уже есть в кэш-памяти, то вместо оригинала считывается копия. В этом случае информация ни в кэш-памяти, ни в основной памяти не изменяется.

Табл.1.

Условия сохранения и обновления информации

Режим работы	Наличие копии ячейки ОП в кэш-памяти	Информация в ячейке кэш-памяти	Информация в ячейке ОП
Чтение	Копия есть		Не изменяется

Не изменяется

Копии нет

Обновляется (создается копия)

Не изменяется

Сквозная запись

Копия есть

Обновляется

Обновляется

Копии нет

Не изменяется

Обновляется

Обратная запись

Копия есть

Обновляется

Не изменяется

Копии нет

Обновляется

Не изменяется

При записи строк существует несколько методов обновления старой информации. Эти методы называются стратегией обновления строк основной памяти. Если результат обновления строк кэш-памяти не возвращается в основную память, то содержимое основной памяти становится неадекватным вычислительному процессу. Чтобы избежать таких ошибок, предусмотрены различные методы обновления основной памяти.
Мультипоточность данных.
Многопоточностью (multithreading) называется способность операционной системы поддерживать в рамках одного процесса выполнение нескольких потоков.

Традиционный подход, при котором каждый процесс представляет собой единый поток выполнения, называется однопоточным подходом. Две левые части рис. 10 иллюстрируют однопоточные подходы. MS DOS является примером операционной системы, способной поддерживать не более одного однопоточного пользовательского процесса. Другие операционные системы, такие, как разнообразные разновидности UNIX, поддерживают процессы множества пользователей, но в каждом из этих процессов может содержаться только один поток. В правой половине рис. 10 представлены многопоточные подходы. Примером системы, в которой один процесс может расщепляться на несколько потоков, является среда выполнения Java. Нас будет интересовать использование нескольких процессов, каждый из которых поддерживает выполнение нескольких потоков. Подобный подход принят в таких операционных системах, как OS/2, Windows 2000 (W2K), Linux, Solaris, Mach, и ряде других.
Рис. 10. Потоки и процессы [ANDE97] Выполнение машинных команд
В многопоточной среде процесс определяется как структурная единица распределения ресурсов, а также структурная единица защиты. С процессами связаны следующие элементы.
Виртуальное адресное пространство, в котором содержится образ процесса.
Защищенный доступ к процессорам, другим процессам (при обмене информацией между ними), файлам и ресурсам ввода-вывода (устройствам и каналам).
В рамках процесса могут находиться один или несколько потоков, каждый из которых обладает следующими характеристиками.
Состояние выполнения потока (выполняющийся, готовый к выполнению и т.д.).
Сохраненный контекст не выполняющегося потока; один из способов рассмотрения потока -- считать его независимым счетчиком команд, работающим в рамках процесса.
Стек выполнения.
Статическая память, выделяемая потоку для локальных переменных.
Доступ к памяти и ресурсам процесса, которому этот поток принадлежит; этот доступ разделяется всеми потоками данного процесса.
На рис. 11 продемонстрировано различие между потоками и процессами с точки зрения управления последними. В однопоточной модели процесса в его представление входит управляющий блок этого процесса и пользовательское адресное пространство, а также стеки ядра и пользователя, с помощью которых осуществляются вызовы процедур и возвраты из них при выполнении процесса. Когда выполнение процесса прерывается, содержимое регистров процессора сохраняется в памяти. В многопоточной среде с каждым процессом тоже связаны управляющий блок и адресное пространство, но теперь для каждого потока создаются свои отдельные стеки, а также свой управляющий блок, в котором содержатся значения регистров, приоритет и другая информация о состоянии потока.
Рис. 11. Однопоточная и многопоточная модели процесса
Таким образом, все потоки процесса разделяют между собой состояние и ресурсы

этого процесса. Они находятся в одном и том же адресном пространстве и имеют доступ к одним и тем же данным. Если один поток изменяет в памяти какие-то данные, то другие потоки во время своего доступа к этим данным имеют возможность отследить эти изменения. Если один поток открывает файл с правом чтения, другие потоки данного процесса тоже могут читать из этого файла.
Перечислим основные преимущества использования потоков с точки зрения производительности:
-Создание нового потока в уже существующем процессе занимает намного меньше времени, чем создание совершенно нового процесса. Исследования, проведенные разработчиками операционной системы Mach, показали, что скорость создания процессов по сравнению с такой же скоростью в UNIX-совместимых приложениях, в которых не используются потоки, возрастает в 10 раз [TEVA87].
-Поток можно завершить намного быстрее, чем процесс.
-Переключение потоков в рамках одного и того же процесса происходит намного быстрее.
-При использовании потоков повышается эффективность обмена информацией между двумя выполняющимися программами. В большинстве операционных систем обмен между независимыми процессами происходит с участием ядра, в функции которого входит обеспечение защиты и механизма, необходимого для осуществления обмена. Однако благодаря тому, что различные потоки одного и того же процесса используют одну и ту же область памяти и одни и те же файлы, они могут обмениваться информацией без участия ядра.
Итак, если приложение или функцию нужно реализовать в виде набора взаимосвязанных модулей, намного эффективнее реализовать ее в виде набора потоков, чем в виде набора отдельных процессов.
Примером приложения, в котором можно удачно применить потоки, является файловый сервер. При получении каждого нового файлового запроса программа управления файлами может порождать новый поток. Из-за того, что серверу приходится обрабатывать очень большое количество запросов, за короткий промежуток времени будут создаваться и удаляться множество потоков. Если такая серверная программа работает на многопроцессорной машине, то на разных процессорах в рамках одного процесса могут одновременно выполняться несколько потоков. Кроме того, из-за того, что процессы или потоки файлового сервера должны совместно использовать данные из файлов, а следовательно, координировать свои действия, рациональнее использовать потоки и общую область памяти, а не процессы и обмен сообщениями.
Потоковая конструкция процесса полезна и на однопроцессорных машинах. Она помогает упростить структуру программы, выполняющей несколько логически различных функций.
В [LETW88] приводится четыре следующих примера использования потоков в однопользовательской многозадачной системе.
Работа в приоритетном и фоновом режимах. В качестве примера можно привести программу электронных таблиц, в которой один из потоков может отвечать за

отображение меню и считывать ввод пользователя, а другой -- выполнять команды пользователя и обновлять таблицу. Такая схема часто увеличивает воспринимаемую пользователем скорость работы приложения, позволяя пользователю начать ввод следующей команды еще до завершения выполнения предыдущей.
Асинхронная обработка. Элементы асинхронности в программе можно реализовать в виде потоков. Например, в качестве меры предосторожности на случай отключения электричества можно сделать так, чтобы текстовый редактор каждую минуту сбрасывал на диск содержимое буфера оперативного запоминающего устройства. Можно создать поток, единственной задачей которого будет создание резервной копии и который будет планировать свою работу непосредственно с помощью операционной системы. Это позволит обойтись без помещения в основную программу замысловатого кода, обеспечивающего проверку соблюдения временного графика или координацию ввода и вывода.
Скорость выполнения. Многопоточный процесс может производить вычисления с одной порцией данных, одновременно считывая с устройства ввода-вывода следующую порцию. В многопроцессорной системе несколько потоков одного и того же процесса могут выполняться одновременно.
Модульная структура программы. Программы, осуществляющие разнообразные действия или выполняющие множество вводов из различных источников и выводов в разные места назначения, легче разрабатывать и реализовывать с помощью потоков.
Планирование и диспетчеризация осуществляются на основе потоков; таким образом, большая часть информации о состоянии процесса, имеющей отношение к его выполнению, поддерживается в структурах данных на уровне потоков. Однако есть несколько действий, которые затрагивают все потоки процесса и которые операционная система должна поддерживать именно на этом уровне. Если процесс приостанавливается, то при этом предполагается, что его адресное пространство будет выгружено из основной памяти. Поскольку все потоки процесса используют одно и то же адресное пространство, все они должны одновременно перейти в состояние приостановленных. Соответственно прекращение процесса приводит к прекращению всех составляющих его потоков.
Сокращенные системы команд
Компьютеры с сокращенным набором команд (КСНК) или в английском варианте RISC-Reduced Instruction Set Computer как направление развития архитектуры ЭВМ воплощает подход, связанный с возвращением к принципам аппаратного управления выполнением команд с целью повышения производительности. Система команд первого и частично второго поколений машин содержали не более пятидесяти команд. Основная проблема, по которой набор команд не расширялся - это цена аппаратуры (и управляющая, и обрабатывающая части процессора реализовывались аппаратно), а также необходимость программирования в кодах (программист не мог запомнить большое количество команд). Период доминирования аппаратного управления: 50-е - начало 60-х годов можно назвать «эра аппаратчиков».

С середины 60-х до 80-х годов доминирует микропрограммное управление выполнением команд, воплощающее «эру программистов», основным лозунгом которой было: «Больше команд хороших и разных!». Этот лозунг соответствовал основным требованиям к процессорам того времени:
Минимизация длины кода программы
вычислительный среда кэш команда
Упрощения реализации компиляторов за счет снижения семантического разрыва между ЯВУ и машинными командами.
Это вызвало рост набора команд компьютеров за счет увеличения их сложности и увеличения числа форматов от 50 до 300 команд (рекордсменом был Vax11/780, у него было 303 команды). Компьютеры с большим набором команд и разнообразием их форматов получили название CISC компьютеров (Complex Instruction Set Computer - машины со сложным набором команд). Для них характерно увеличение сложности и соответственно размеров микропрограммного устройства управления, которое интерпретировало выполнение этих команд. К этому времени благодаря технологическим достижениям тактовая частота процессоров стала достигать 100Мгц и повышение производительности требовало размещения всех частей процессора на одном кристалле для сокращения длины соединений его элементов. В то же время микропрограммное управление из-за своей сложности стало занимать до шестидесяти процентов площади кристалла, что либо не допускало использования эффективных средств арифметической обработки данных, либо требовало размещения частей процессора на разных кристаллах. Все это приводило к существенному ограничению производительности, увеличивало сроки разработки и снижало выход годных кристаллов.
В 80-х годах рядом исследователей было замечено, что при выполнении большинства программ наиболее активно используется около 30% сравнительно простых команд арифметики и управления. Постепенно стало формироваться направление развития архитектуры компьютеров, требующее чтобы система команд процессора содержала минимальный набор наиболее часто используемых и наиболее простых команд (возврат к примерно 50 командам). Это направление получило название - КСНК (компьютеры с сокращенным набором команд) или RISC (Reduced Instruction Set Computer) - и имеет лозунг: «Меньше команд, выше скорость выполнения!». В результате в конце 80-х г.г. благодаря развитию технологии производства СБИС и их удешевлению, а также развитию методов и опыта разработки оптимизирующих компиляторов постепенно сложились основные принципы (или законы) RISC-архитектур:
Основной набор команд не должен интерпретироваться микрокомандами, а должен выполняться аппаратным обеспечением. Все команды должны иметь одинаковую длину и минимальное число форматов (обычно не более 2-3), это упрощает логику управления при выборе и при исполнении команды. Любая команда основного набора должна выполняться за один машинный цикл, обратно пропорциональный тактовой частоте процессора (стандартом является команда сложения регистра с регистром, занимающая от 3 - 10нс.); это достигается одновременным

(параллельным) выполнением максимально возможного числа команд путем конвейеризации или использования нескольких обрабатывающих узлов. Обращение к памяти производиться только по специально выделенным командам работы с памятью типа: Load - загрузка и Store - сохранение, а вся обработка данных должна вестись в регистровом формате; при этом количество регистров должно быть велико (100 и более). Система команд должна обеспечивать поддержку компиляции с конкретного языка программирования (компиляторы для RISC на порядок сложнее, чем компиляторы для CISC).

Список литературы

1. Гуда А.Н., Бутакова М.А., Нечитайло Н.М., Чернов А.В. Информатитка. Общий курс: Учебник / Под ред. академика РАН В.И. Колесникова. - М.: Издательско-торговая корпорация «Дашков и К». Ростов н / Д : Наука - Пресс, 2008. - 400с.
2. Информатика: Базовый курс: / О.А. Акулов, Н.В.Медведев. 2-е изд., испр. и доп. - М.: Омега - Л, 2005.- 552с.
3. Информатика: Учебник. - 3-е перераб. изд. / Под ред. Н.В. Макаровой. - М.: Финансы и статистика, 2002. - 768с.: ил.
4. Информатика. Базовый курс / Симонович С.В. и др. - Спб: Издательство «Питер», 2000. - 640с.: ил.
5. Водяхо А.И., Горнец Н.Н., Пузанков Д.В. Высокопроизводительные системы обработки данных: Учеб. пособие для вузов. - М.: Высш. шк., 1997.
6. Компьютерные системы и сети. Учеб. пособие /В.П. Косарев и др. - М.: Финансы и статистика, 1999....