НУКУССКИЙ ФИЛИАЛ ТАШКЕНТСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИИ ИМЕНИ МУХАММАДА АЛ-ХОРАЗМИЙ

КУРС ЛЕКЦИЙ ПО ПРЕДМЕТУ

СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ. ВНЕШНИЕ СИСТЕМЫ.

1-часть

Составил: **Ф.М.Зарипов**

Нукус - 2019

Оглавление

ГЛАВА-I. ВВЕДЕНИЕ. ОСНОВНЫЕ ПОНЯТИЯ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ

Лекция 1.1. Основные понятия систем реального времени. Структура, состав и основные технические и технологические показатели систем реального времени.

План

- Основные понятия
- Структура СРВ
- ОСРВ и ОС общего назначения
- Механизмы реального времени
- Часы и таймеры

Основные понятия

Существует несколько определений систем реального времени (СРВ), большинство из которых противоречат друг другу. Приведем несколько из них, чтобы продемонстрировать различные взгляды на назначение и основные задачи СРВ.

1. Системой реального времени называется система, в которой успешность работы любой программы зависит не только от ее логической правильности, но от времени, за которое она получила результат. Если временные ограничения не удовлетворены, то фиксируется сбой в работе системы.

   Таким образом, временные ограничения должны быть гарантировано удовлетворены. Это требует от системы быть предсказуемой, т.е. вне зависимости от своего текущего состояния и загруженности выдавать нужный результат за требуемое время. При этом желательно, чтобы система обеспечивала как можно больший процент использования имеющихся ресурсов.
2. Стандарт POSIX 1003.1 определяет СРВ следующим образом: «Реальное время в операционных системах – это способность операционной системы обеспечить требуемый уровень сервиса в заданный промежуток времени».
3. Иногда системами реального времени называется системы постоянной готовности (on-line системы), или «интерактивные системы с достаточным временем реакции». Обычно это делают по маркетинговым соображениям. Действительно, если интерактивную программу называют работающей в «реальном времени», то это просто означает, что она успевает обработать запросы от человека, для которого задержка в сотни миллисекунд даже незаметна.
4. Иногда понятие «система реального времени» отождествляют с понятием «быстрая система». Это не всегда правильно. Время задержки СРВ на событие не так уж важно (оно может достигать нескольких секунд). Главное, чтобы это время было достаточно для рассматриваемого приложения и *гарантировано*. Очень часто алгоритм с гарантированным временем работы менее эффективен, чем алгоритм, таким действием не обладающий. Например, алгоритм «быстрой» сортировки в среднем работает значительно быстрее других алгоритмов сортировки, но его гарантированная оценка сложности значительно хуже.

5. Во многих сферах приложения СРВ вводят свои понятия «реального времени». Например, процесс цифровой обработки сигнала называют идущим в «реальном времени», если анализ (при вводе) и/или генерация (при выводе) данных может быть проведен за то же время, что и анализ и/или генерация тех же данных без цифровой обработки сигнала. Например, если при обработке аудио данных требуется 2.01 секунд для анализа 2.00 секунд звука, то это не процесс реального времени. Если же требуется 1.99 секунд, то это процесс реального времени.

Назовем системой реального времени аппаратно-программный комплекс, реагирующий в предсказуемые времена на непредсказуемый поток внешних событий

Это определение означает, что:

- Система должна успеть отреагировать на событие, произошедшее на объекте, в течение времени, критического для этого события (meet deadline). Величина критического времени для каждого события определяется объектом и самим событием, и, естественно, может быть разной, но время реакции системы должно быть предсказано (вычислено) при создании системы. Отсутствие реакции в предсказанное время считается ошибкой для систем реального времени.
- Система должна успевать реагировать на одновременно происходящие события. Даже если два или больше внешних событий происходят одновременно, система должна успеть среагировать на каждое из них в течение интервалов времени, критического для этих событий.

Хорошим примером задачи, где требуется СРВ, является управление роботом, берущим деталь с ленты конвейера. Деталь движется, и робот имеет лишь маленькое временное окно, когда он может ее взять. Если он опоздает, то деталь уже не будет на нужном участке конвейера, и, следовательно, работа не будет сделана, несмотря на то, что робот находится в правильном месте.

Если он позиционируется раньше, то деталь еще не успеет подъехать, и робот заблокирует ей путь. Другим примером может быть самолет, находящийся на автопилоте. Сенсорные серводатчики должны постоянно передавать в управляющий компьютер результаты измерений. Если результат какого-либо измерения будет пропущен, то это может привести к недопустимому несоответствию между реальным состоянием систем самолета и информацией о нем в управляющей программе.

Различают системы реального времени двух типов - системы жесткого реального времени и системы мягкого реального времени.

Системы жесткого реального времени не допускают никаких задержек реакции системы ни при каких условиях, так как:

- результаты могут оказаться бесполезны в случае опоздания,
- может произойти катастрофа в случае задержки реакции,
- стоимость опоздания может оказаться бесконечно велика.

Примеры систем жесткого реального времени - бортовые системы управления, системы аварийной защиты, регистраторы аварийных событий.

Системы мягкого реального времени характеризуются тем, что задержка реакции не критична, хотя и может привести к увеличению стоимости результатов и снижению производительности системы в целом.

Пример - работа сети. Если система не успела обработать очередной принятый пакет, это приведет к таймауту на передающей стороне и повторной посылке (в зависимости от протокола, конечно). Данные при этом не теряются, но производительность сети снижается.

Основное отличие между системами жесткого и мягкого реального времени можно выразить так: система жесткого реального времени никогда не опоздает с реакцией на событие, система мягкого реального времени - не должна опаздывать с реакцией на событие.

Структура СРВ

Любая система реального масштаба времени может быть описана как состоящая из трех основных подсистем.

Управляемая (контролируемая) подсистема (например, индустриальный завод, управляемое компьютером транспортное средство), диктует требования в реальном масштабе времени; **подсистема контроля (контролирующая)** управляет некоторыми вычислениями и связью с оборудованием для использования от управляемой подсистемы; **подсистема оператора (операционная)** контролирует полную деятельность системы.

Интерфейс между управляемыми и подсистемами контроля состоит из таких устройств как датчики и приводы. Интерфейс между управляющей подсистемой и оператором связывает человека с машинной.

Управляемая подсистема представлена задачами (в дальнейшем называемыми прикладными задачами) которые используют оборудование, управляемое подсистемой контроля. Эта последняя подсистема может быть построена из очень большого количества процессоров, управляющими такими местными ресурсами, как память и устройства хранения, и доступ к локальной сети в реальном масштабе времени. Эти процессоры и ресурсы управляются системой программного обеспечения, которую мы называем **операционной системой реального масштаба времени** (RTOS – real time operating system).

ОСРВ и ОС общего назначения

Назовем **операционной системой реального времени** такую систему, которая может быть использована для построения систем жесткого реального времени.

Это определение выражает отношение к операционным системам реального времени как к объекту, содержащему необходимые инструменты, но также означает, что этими инструментами еще необходимо правильно воспользоваться.

Большинство программного обеспечения ориентировано на «слабое» реальное время, а задача СРВ – обеспечить уровень безопасного функционирования системы, даже если управляющая программа никогда не закончит своей работы.

ОС общего назначения, особенно многопользовательские, такие как UNIX, ориентированы на оптимальное распределение ресурсов компьютера между пользователями и задачами (системы разделения времени), В операционных системах реального времени подобная задача отходит на второй план - все отступает перед главной задачей - успеть среагировать на события, происходящие на объекте.

Другое отличие - применение операционной системы реального времени всегда связано с аппаратурой, с объектом, с событиями, происходящими на объекте. Система реального времени, как аппаратно-программный комплекс, включает в себя датчики, регистрирующие события на объекте, модули ввода-вывода, преобразующие показания датчиков в цифровой вид, пригодный для обработки этих показаний на компьютере, и, наконец, компьютер с программой, реагирующей на события, происходящие на объекте. Операционная система реального времени ориентирована на обработку внешних событий. Именно это приводит к коренным отличиям (по сравнению с ОС общего назначения) в структуре системы, в функциях ядра, в построении системы ввода-вывода.

Операционная система реального времени может быть похожа по пользовательскому интерфейсу на ОС общего назначения (к этому, кстати, стремятся почти все производители операционных системах реального времени), однако устроена она совершенно иначе - об этом речь впереди. Кроме того, применение операционных системах реального времени всегда конкретно. Если ОС общего назначения обычно воспринимается пользователями (не разработчиками) как уже готовый набор приложений, то операционная система реального времени служит только инструментом для создания конкретного аппаратно-программного комплекса реального времени. И поэтому наиболее широкий класс пользователей операционных системах реального времени - разработчики комплексов реального времени, люди, проектирующие системы управления и сбора данных. Проектируя и разрабатывая конкретную систему реального времени, программист всегда точно знает, какие события могут произойти на объекте, знает критические сроки обслуживания каждого из этих событий. Одно из коренных внешних отличий систем реального времени от систем общего назначения - четкое разграничение систем разработки и систем исполнения.

Система исполнения операционных системах реального времени - набор инструментов (ядро, драйверы, исполняемые модули), обеспечивающих функционирование приложения реального времени. Большинство современных ведущих операционных систем реального времени поддерживают целый спектр аппаратных архитектур, на которых работают системы исполнения (Intel, Motorola, RISC,MIPS, PowerPC, и другие). Это объясняется тем, что набор аппаратных средств - часть комплекса реального времени и аппаратура должна быть также адекватна решаемой задаче, поэтому ведущие операционные системы реального времени перекрывают целый ряд наиболее популярных архитектур, чтобы удовлетворить самым разным требованиям по части аппаратуры.

Система исполнения операционных системах реального времени и компьютер, на котором она исполняется называют **"целевой"** (target) системой. Система разработки - набор средств, обеспечивающих создание и отладку приложения реального времени.

Системы разработки (компиляторы, отладчики и всевозможные tools) работают, как правило, в популярных и распространенных ОС, таких, как UNIX и Windows. Кроме того, многие операционные системы реального времени имеют и так называемые резидентные средства разработки, исполняющиеся в среде самой операционной системы реального времени. Заметим, что функционально средства разработки операционных систем реального времени отличаются от привычных систем разработки, таких, например, как Developers Studio, TaskBuilder, так как часто они содержат средства удаленной отладки, средства профилирования (измерение времен выполнения отдельных участков кода), средства эмуляции целевого процессора, специальные средства отладки взаимодействующих задач, а иногда и средства моделирования.

Характеристики ОСРВ

Как было сказано ранее, сердцем системы реального времени является ОСРВ (операционная система реального времени). В связи со специфичность решаемых задач, ОСРВ должна обладать определенными свойствами.

Время реакции системы на внешние события. Согласно определению, ОСРВ должна обеспечить требуемый уровень сервиса в заданный промежуток времени. Этот промежуток времени задается обычно периодичностью и скоростью процессов, которым управляет система.

Приблизительное время реакции в зависимости от области применения ОСРВ может быть следующее:

- математическое моделирование - несколько **микросекунд**
- радиолокация - несколько **миллисекунд**
- складской учет - несколько **секунд**
- управление производством - несколько **минут**

Видно, что времена очень разнятся и накладывают различные требования на вычислительную установку, на которой работает ОСРВ.

Итак, как же определить время реакции системы? Поймем, какие времена мы должны знать для того, чтобы предсказать время реакции системы. События, происходящие на объекте, регистрируются датчиками, данные с датчиков передаются в модули ввода-вывода (интерфейсы) системы. Модули ввода-вывода, получив информацию от датчиков и преобразовав ее, генерируют запрос на прерывание в управляющем компьютере, подавая ему тем самым сигнал о том, что на объекте произошло событие. Получив сигнал от модуля ввода-вывода, система должна запустить программу обработки этого события. Интервал времени - от события на объекте и до выполнения первой инструкции в программе обработки этого события и является **временем реакции системы на события**, и, проектируя систему реального времени, разработчики должны уметь вычислять этот интервал. Из чего он складывается? Время выполнения цепочки действий - от события на объекте до генерации прерывания - никак не зависит от операционных систем реального времени и целиком определяется аппаратурой, а вот интервал времени - от возникновения запроса на прерывание и до выполнения первой инструкции обработчика определяется целиком свойствами операционной системы и архитектурой компьютера. Причем это время нужно уметь оценивать в худшей для системы ситуации, то есть в предположении, что процессор загружен, что в это время могут происходить другие прерывания, что система может выполнять какие-то действия, блокирующие прерывания. В ОСРВ заложен параллелизм, возможность одновременной обработки нескольких событий, поэтому все операционные системы реального времени являются многозадачными (многопроцессными, многонитиевыми). Для того, чтобы уметь оценивать накладные расходы системы при обработке параллельных событий, необходимо знать время, которое система затрачивает на передачу управления от процесса к процессу (от задачи к задаче, от нити к нити), то есть **время переключения контекста**.

ОСРВ содержат механизмы, гарантирующие заранее вычисленное время реакции системы.

Эта гарантия достигается знанием максимального времени блокировок прерываний в системе, времени переключения контекста, времен выполнения различных системных вызовов, применением нужных механизмов диспетчеризации и пр. Т.е. время реакции на события для операционных систем реального времени можно вычислить с большой точностью. Эти вычисления невозможны для операционных систем LINUX и Windows NT - здесь можно полагаться только на результаты тестирования, эмпирические оценки. Часто производительность компьютера подбирается таким образом, чтобы успевать в нужные времена, но сути дела это не меняет, поэтому риски в сложных комплексах могут оказаться велики - при небольшом изменении внешних условий или при модификации приложений времена могут "поплыть". Что касается расширений реального времени для LINUX и Windows NT, то здесь, по крайней мере, можно полагаться на времена, полученные при тестировании.

Время перезагрузки системы. Этот параметр кажется второстепенным, однако были свидетелями случаи, когда именно этот параметр целиком определял выбор дорогой операционной системы (время перезагрузки не должно было превышать 1 секунду). Этот параметр важен для систем, от которых требуется непрерывная работа; в этих случаях ставятся ловушки, отслеживающие зависание системы или приложений, и, если таковое произошло, автоматически перезагружающие систему (такие ловушки необходимы в системах повышенной надежности, т.к. от ошибок, по крайней мере, в приложениях никто не застрахован). В таких случая важным является такое свойство системы как ее живучесть при незапланированных перезагрузках. Большинство операционных систем реального времени устойчивы к перезагрузкам и могут быть прерваны и перезагружены в любое время.

Время загрузки для разных ОСРВ колеблется от секунды до нескольких десятков секунд. В большинстве систем (OS9, VxWorks) время загрузки можно регулировать, изменяя стартовые последовательности. В ОС LINUX время _____загрузки в стандартном варианте более минуты, система неустойчива к внезапным остановам - требуется стандартная процедура завершения работы с системой (shutdown). Однако LINUX достаточно гибок и можно создать конфигурации системы, в которых время загрузки будет уменьшено до десятка секунд и система будет устойчива к сбоям (использование специальной опции файловой системы). В Windows NT время загрузки более минуты, система неустойчива к внезапным сбоям. Использование расширений реального времени (RTX) позволяет детектировать зависания системы и выполнить в этих случаях необходимые операции по спасению данных и по выполнению каких-то страховочных действий.

Посмотрим, какие ОС могут использоваться в системах реального времени в зависимости от времени реакции системы.

Вычислительные установки, на которых применяются ОСРВ, можно разделить на три группы:

- **«Обычные» компьютеры.** По логическому устройству совпадают с настольными системами. Аппаратное устройство несколько отличается. Для обеспечения минимального времени простоя в случае технической неполадки процессор, память и т.д. размещаются на съемной плате, вставляемой в специальный разъем так называемой «пассивной» основной платы. В другие разъемы этой платы

вставляются платы периферийных устройств. Среди процессоров таких компьютеров преобладают процессоры Intel.

- **Промышленные компьютеры.** Состоят из одной платы, на которой размещены процессор, контроллер памяти, память. Память может быть нескольких видов – ПЗУ (в которой размещается сама ОСРВ), ОЗУ (там размещаются код и данные), Флеш-память (может играть роль диска). На плате также находятся контроллеры периферийных устройств, несколько программируемых таймеров. Среди процессоров этих компьютеров доминируют процессоры семейства PowerPC, Motorola 68xxx, SPARC, ARM, Intel 80x86, 80960x.
- **Встраиваемые системы.** Устанавливаются внутрь оборудования, которым они управляют. Для крупного оборудования могут по исполнению совпадать с промышленными компьютерами. Для оборудования поменьше могут представлять собой процессор с сопутствующими элементами, размещенными на одной плате с другими электронными компонентами оборудования. Для миниатюрных систем процессор с сопутствующими элементами может быть частью одной из интегральных схем оборудования.

В связи с особенностями оборудования ОСРВ (промышленные компьютеры и встраиваемые системы часто являются бездисковыми.) должны обладать следующими свойствами:

Размеры системы. Для систем реального времени важным параметром является размер системы исполнения, а именно суммарный размер минимально необходимого для работы приложения системного набора (ядро, системные модули, драйверы и т. д.). Хотя, надо признать, что с течением времени значение этого параметра уменьшается, тем не менее он остается важным и производители систем реального времени стремятся к тому, чтобы размеры ядра и обслуживающих модулей системы были невелики.

Примеры: размер ядра операционной системы реального времени OS-9 на микропроцессорах Motorola 68xxx - 22 KB, VxWorks - 16 KB.

Возможность исполнения системы из ПЗУ (ROM). Система должна иметь возможность осуществлять загрузку из ПЗУ. Для экономии места в ПЗУ часть системы может храниться в сжатом виде и загружаться в ОЗУ по мере необходимости. Часто система позволяет исполнять код как в ПЗУ, так и в ОЗУ. При наличии свободного места в ОЗУ система может копировать себя из медленного ПЗУ в более быстрое ОЗУ.

К дополнительным свойствам ОСРВ можно отнести следующие:

Наличие необходимых драйверов устройств. Если разрабатываемая система имеет обширную периферию, то наличие уже готовых драйверов может оказать большое влияние на выбор операционной системы. Естественно, самый большой набор драйверов создан для операционных системах LINUX и Windows NT. Наиболее популярные операционные системы реального времени, такие как VxWorks, OS9, QNX, также имеют обширные наборы готовых драйверов и, кроме того, содержат средства для их быстрой разработки.

Поддержка процессоров различной архитектуры. В связи с тем, что в промышленных компьютерах, серверах, встраиваемых системах широко распространены процессоры разной архитектуры с различной системой команд, ОСРВ по возможности должна поддерживать как можно более широкий ряд процессоров.

Одной из важных характеристик ОСРВ является наличие **специального кроссплатформенного инструментария разработчика.** Это связано с тем, что разработка СРВ часто проводится на «обычном» компьютере, отличном по архитектуре от компьютера, на котором будет устанавливаться СРВ. При этом ОС на этих двух компьютерах также может не совпадать.

Механизмы реального времени

Важнейшими характеристиками ОСРВ являются заложенные в операционную систему **механизмы реального времени.**

Процесс проектирования конкретной системы реального времени начинается с тщательного изучения объекта. Разработчики проекта исследуют объект, изучают возможные события на нем, определяют критические сроки реакции системы на каждое событие и разрабатывают алгоритмы обработки этих событий. Затем следует процесс проектирования и разработки программных приложений. Какие же механизмы в операционных системах реального времени делают систему реального времени (СРВ) предсказуемой?

Система приоритетов и алгоритмы диспетчеризации. Базовыми инструментами разработки сценария работы системы являются система приоритетов процессов (задач) и алгоритмы планирования (диспетчеризации) операционных системах реального времени.

В многозадачных ОС общего назначения используются, как правило, различные модификации алгоритма круговой диспетчеризации, основанные на понятии непрерывного кванта времени ("time slice"), предоставляемого процессу для работы. Планировщик по истечении каждого кванта времени просматривает очередь активных процессов и принимает решение, кому передать управление, основываясь на приоритетах процессов (численных значениях, им присвоенных).

Приоритеты могут быть фиксированными или меняться со временем - это зависит от алгоритмов планирования в данной ОС, но рано или поздно процессорное время получат все процессы в системе.

Алгоритмы круговой диспетчеризации неприменимы в чистом виде в операционных системах реального времени. Основной недостаток - непрерывный квант времени, в течение которого процессором владеет только один процесс. Планировщики же операционных систем реального времени имеют возможность сменить процесс до истечения "time slice", если в этом возникла необходимость. Один из возможных алгоритмов планирования при этом "приоритетный с вытеснением". Мир операционных систем реального времени отличается богатством различных алгоритмов планирования: динамические, приоритетные, монотонные, адаптивные и пр., цель же всегда преследуется одна - предоставить инструмент, позволяющий в нужный момент времени исполнять именно тот процесс, который необходим.

Механизмы межзадачного взаимодействия. Другой набор механизмов реального времени относится к средствам синхронизации процессов и передачи данных между ними. Для операционных систем реального времени характерна развитость этих механизмов. К таким механизмам относятся: семафоры, мьютексы, события, сигналы, средства для работы с разделяемой памятью, каналы данных (pipes), очереди сообщений. Многие из подобных механизмов используются и в ОС общего назначения, но их реализация в

операционных системах реального времени имеет свои особенности - время исполнения системных вызовов почти не зависит от состояния системы, и в каждой операционной системе реального времени есть по крайней мере один быстрый механизм передачи данных от процесса к процессу.

Средства для работы с таймерами. Такие инструменты, как средства работы с таймерами, необходимы для систем с жестким временным регламентом, поэтому развитость средств работы с таймерами - необходимый атрибут операционных систем реального времени. Эти средства, как правило, позволяют:

- измерять и задавать различные промежутки времени (от 1 мкс и выше),
- генерировать прерывания по истечении временных интервалов,
- создавать разовые и циклические будильники

Здесь описаны только базовые, обязательные механизмы, использующиеся в ОСРВ. Кроме того, почти в каждой операционной системе реального времени имеется целый набор дополнительных, специфических только для нее механизмов, касающийся системы ввода-вывода, управления прерываниями, работы с памятью. Каждая система содержит также ряд средств, обеспечивающих ее надежность. Позже механизмы реального времени будут рассмотрены подробнее.

Часы и таймеры

В ОСРВ используются различные службы времени. Операционная система отслеживает текущее время, в определенное время запускает задачи и потоки и приостанавливает их на определенные интервалы. В службах времени ОСРВ используются часы реального времени. Обычно используются высокоточные аппаратные часы. Для отсчета временных интервалов на основе часов реального времени создаются таймеры. Для каждого процесса и потока определяются часы процессорного времени. На базе этих часов создаются таймеры; которые измеряют перерасход времени процессом или потоком, позволяя динамически выявлять программные ошибки или ошибки вычисления максимально возможного времени выполнения. В высоконадежных, критичных ко времени системах важно выявление ситуаций, при которых задача превышает максимально возможное время своего выполнения, т.к. при этом работа системы может выйти за рамки допустимого времени отклика. Часы времени выполнения позволяют выявить возникновение перерасхода времени и активизировать соответствующие действия по обработке ошибок. Большинство ОСРВ оперируют относительным временем. Что-то происходит “до” и “после” некоторого другого события. В системе, полностью управляемой событиями, необходим часовой механизм (ticker), т.к. там нет квантования времени (time slicing). Однако, если нужны временные метки для некоторых событий или необходим системный вызов типа “ждать одну секунду”, то нужен тактовый генератор и/или таймер. Синхронизация в ОСРВ осуществляется с помощью механизма блокирования (или ожидания) до наступления некоторого события. Абсолютное время не используется. Реализации в ОСРВ других концептуальных абстракций подобны их реализациям в традиционных ОС.

ГЛАВА-II. КОНТРОЛЬ И УПРАВЛЕНИЯ В СИСТЕМАХ РЕАЛЬНОГО ВРЕМЕНИ

Лекция 2.1. Задачи контроля и управления, типы объектов, обработка и порядок работы управления систем реального времени.

План

- Управление как процесс принятия и реализации решений
- Классификация систем управления
- Структура систем реального времени

Под системой принято понимать совокупность связанных между собой элементов. Система характеризуется структурой S = = <E, C, λ>, где E – множество элементов, C – множество связей между элементами, λ – соответствие между элементами. При этом элементом называется часть системы, не подлежащая дальнейшему делению на части при данном уровне рассмотрения. Элементы могут представлять собой понятия, и тогда мы будем иметь дело с понятийной системой; если элементами системы являются физические объекты, то налицо техническая система. Заметим, что в определении отмечается некоторая совокупность, т.е. некоторая целостность. Нас интересуют не отдельные элементы, а все вместе, взятые в некотором единстве. Принципиальным является необходимость учета взаимодействия элементов, так как именно оно придает совокупности элементов новое качество. Часть элементов системы, выделенная на основе какой-либо общности элементов, называется подсистемой. Системы, создаваемые человеком, обладают целью функционирования. Именно наличие цели служит тем объединяющим фактором, на основе которого объединяются элементы. Такие системы называют организованными системами.

Управление как процесс принятия и реализации решений

В повседневной деятельности мы часто сталкиваемся с терминами «управлять», «регулировать» и понимаем под этими терминами задачу воздействия на параметры объекта для достижения определенных целей. Естественно называть управлением целенаправленное изменение параметров объекта управления. Совокупность наиболее важных параметров, от которых зависит функционирование объекта управления, описывает его состояние. Знание состояния объекта управления – важнейший фактор, определяющий принятие решения по управлению. Другой важный фактор – возмущения, которым подвергается объект управления со стороны других объектов, состоянием которых мы не можем управлять. Множество таких объектов объединяется понятием «внешняя среда». На основе оценок состояния объекта управления и внешней среды принимается решение по управлению. Конечно, при этом учитывается цель функционирования и ресурсы, которыми мы располагаем. После того, как решение приято, оно реализуется с потреблением ресурсов. Так мы приходим к структурной схеме процесса управления.

Управление осуществляется путем выдачи на объект управляющих воздействий

$$X(t) = (x1, x2,\ldots, xn; t).$$

Кроме них, объект управления подвергается воздействию возмущений со стороны внешней среды; часть из них

$$Z(t) = (z1, z2,\ldots, zm; t)$$

измеряется, другая часть

$$F(t) = (f1, f2, \ldots, fs; t)$$

неизвестна (часто неизвестно также значение s). Контролируемые параметры, по которым ведется управление,

$$Y(t) = (y1, y2, \ldots, yr; t),$$

называются управляемыми. Управляемые параметры дают обобщенную информацию о состоянии объекта управления

$$S(t) = (s1, s2,\ldots,sk;t).$$

Смена состояний объекта управления определяет процесс его функционирования:

$$S(t) = \phi[X(t), Z(t), F(t), Y(t), S(t-1)].$$

Задача управления состоит в том, чтобы найти такую функцию X(t), которая обеспечила бы перевод объекта в заданное конечное состояние S*, соответствующее цели его функционирования. Этот перевод должен закончиться за конечное число шагов и требовать минимальных затрат ресурсов R.

Классификация систем управления

Тип системы зависит от того, как принимается решение по управлению. Если в процессе принятия решения участвует человек, то такая система называется автоматизированной (АСУ); если человек исключен из процесса принятия решения, то это – автоматическая система управления (САУ). По физической природе объекта управления различают системы управления технологическими процессами, где объект управления представляет собой физический объект, и системы управления экономико-организационного типа, в которых объектом управления является хозяйственная деятельность. В последнее время создаются интегрированные системы управления, в которых присутствуют объекты обоих типов. Другим признаком, по которому классифицируются системы, является время принятия решения. В системах управления оно зависит от динамики объекта, которая может быть различной: от миллисекунд до часов и дней. Условная граница лежит на уровне 100 мс: если время принятия решения не превышает этой величины, то говорят, что система функционирует в реальном масштабе времени; если превышает, то говорят о произвольном масштабе. Системы управления технологическими процессами всегда функционируют в реальном масштабе времени. Время в таких системах является одним из факторов, определяющих эффективность системы.

Хорошим примером системы реального времени является промышленный робот, который должен брать что-то с ленты конвейера. Объекты на конвейере движутся, и робот имеет некоторый интервал времени для того, чтобы схватить объект. Если робот опоздает, то объекта уже не будет на месте и поэтому работа будет неверной. Если робот поспешит, то объекта там еще не будет. Принято различать системы «жесткого» и «мягкого» реального времени. Системой «жесткого» реального времени называют систему, в которой

неспособность обеспечить реакцию на какие-либо события в заданное время является отказом и ведет к невозможности решения поставленной задачи. В качестве условной временной границы допустимого времени реакции обычно принимают 100 мкс. В жесткой системе: - никакое опоздание неприемлемо ни при каких обстоятельствах; - результат, выданный с опозданием, бесполезен; - нарушение крайнего срока времени отклика рассматривается как катастрофический отказ; - цена превышения заданного времени отклика бесконечно велика. Точного определения для «мягкого» реального времени не существует, но принято считать, что она иногда может не успевать делать все, что надо, в установленные сроки. В мягкой системе: - возрастает цена за опоздание результата; - критическим фактором является низкая производительность, а не опоздания.

В терминах вероятностей эти определения могут быть записаны следующим образом:

$$P\{|t - t0| > \Delta t\} > 0$$

– для систем «мягкого» реального времени,

$$P\{|t - t0| > \Delta t\} = 0$$

– для систем «жесткого» реального времени (здесь t – фактический момент выдачи управляющего воздействия, t0 – заданный момент выдачи управляющего воздействия, Δt – допустимая погрешность).

В зависимости от функций, выполняемых системой управления, различают виды (формы) автоматизации: управление, регулирование, контроль, защиту и блокировку.

Управление представляет собой принятие решений и выдачу на объект управления совокупности воздействий, выполняемых на основании информации о ходе технологического процесса, с целью поддержания требуемого технологического режима или улучшения его параметров. Обычно предполагается, что управление подразумевает поиск и реализацию оптимального (по определенному критерию) решения.

Регулирование представляет собой принятие решений и выдачу на объект управления совокупности воздействий, выполняемых на основании информации о ходе технологического процесса, с целью поддержания параметров производственных процессов постоянными или изменяющимися по заданному закону.

Контроль – это наблюдение за параметрами объекта управления с целью определения его состояния. Параметры, подлежащие контролю, в зависимости от их физической природы различны (температура, давление, расход топлива, число оборотов, сила тока и т.п.). Контроль может быть местным и дистанционным. Местный контроль дает возможность наблюдать за состоянием параметра непосредственно в контролируемой точке. При дистанционном контроле за состоянием параметров можно следить на расстоянии от контролируемой точки.

Защита – это комплекс средств и мероприятий по предохранению агрегатов и установок при нарушениях технологических режимов.

Блокировка – это комплекс средств и мероприятий, предохраняющих участок установки или агрегат от неправильных операций вследствие невнимательности оператора или ошибочных команд. Различают две группы блокировки: запретно-разрешающую и аварийную.

Запретно-разрешающие блокировки предотвращают неправильные включения и отключения механизмов, а также нарушение установленной технологическими требованиями очередности пуска и остановки различных механизмов.

Аварийные блокировки предназначены для автоматического последовательного отключения (включения) механизмов в соответствии с режимом работы агрегата, подвергшегося аварийному отключению. Блокировочные устройства применяют также в случаях, когда требуется исключить одновременную подачу команд противоположного знака на один и тот же регулирующий орган из разных мест или от автоматического и дистанционного управления, когда оба вида управления действуют параллельно.

Структура систем реального времени

Основными составляющими автоматизированных систем реального времени (АСУ ВР) являются комплекс технических средств, программное и информационное обеспечение. Для принятия решения по управлению нужно знать о состоянии объекта управления и возмущениях со стороны внешней среды. Эта информация вводится в систему с помощью измерительных средств, которые являются источниками информации для средств переработки информации. Эти средства реализуют алгоритм принятия решения и формируют управляющие воздействия, которые передаются на исполнительные средства, непосредственно связанные с изменяемыми параметрами объекта управления. Для того чтобы человек участвовал в процессе принятия решения, ему также нужно дать возможность следить за состоянием объекта и внешней среды. Для этого он снабжается устройствами отображения информации, которые предназначены для преобразования информации в удобную для восприятия форму (чаще всего – в визуальную). Человек может вмешиваться в управление, воздействуя на исполнительные средства и на средства переработки информации с помощью управляющих средств. Поскольку части системы могут находиться на значительных расстояниях друг от друга, часть связей представляет собой средства передачи данных – специальные технические устройства, предназначенные для этих целей. Еще одна составляющая комплекса технических средств – система энергоснабжения; ее назначение – снабжение всех технических средств электроэнергией заданного напряжения и мощности.

В качестве средств переработки информации обычно применяется узел на основе вычислительной машины. Тип и структура этого узла может быть различной – от простейшего контроллера до кластера, однако в любом случае его применение связано со следующими особенностями: – он должен «следить» за множеством параллельно протекающих процессов; – он должен обрабатывать запросы, поступающие в произвольные моменты времени; – допустимое время принятия решения обычно соизмеримо с временем реализации алгоритма выработки такого решения; – он должен удовлетворять повышенным требованиям по надежности и достоверности информации; – состав задач, которые решает этот узел, заранее известен и программное обеспечение для их решения отлажено. Эти особенности учитываются при построении программного обеспечения. Оно включает в себя операционную систему, программы решения функциональных задач и программы контроля и обеспечения устойчивости вычислительного процесса.

Главное назначение операционной системы – обеспечение параллелизма и обработка заявок, поступающих в случайные моменты времени. Программы решения

функциональных задач выполняют основное целевое назначение системы – управление объектом. Третья группа программ контролирует работу системы. Информационное обеспечение содержит данные, необходимые для управления системой.

ГЛАВА-III. СРЕДСТВА ПЕРЕРАБОТКИ ИНФОРМАЦИИ

ЛЕКЦИЯ 3.1. КЛАССИФИКАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ. МНОГОМАШИННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

План

- Классификация ВС
- Многомашинные вычислительные системы
- Уровни комплексирования

Первые вычислительные системы включали большие и дорогие ЭВМ, в связи с чем их количество в такой системе не превышало трех – четырех. Появление персональных компьютеров привело к построению распределенных систем управления, в которые могут быть вовлечены десятки и сотни компьютеров – персональных рабочих станций.

Эффективность использования вычислительных систем во многом определяется возможностями организации параллельных вычислений на уровне алгоритмов и программ.

Когда параллелизм реализуется внутри процессора (с помощью, например, конвейерного АЛУ или нескольких АЛУ), то такая вычислительная структура определяется как *векторный* или *суперскалярный* процессор.

Если параллелизм обеспечивается путем объединения нескольких ЭВМ или процессоров, то такая вычислительная структура определяется как *вычислительная система*.

Вычислительная система, в которой отдельные ЭВМ или процессоры объединяются вместе посредством специального коммуникационного оборудования (каналов связи), представляет собой *вычислительную сеть*.

Отличительной особенностью вычислительной системы по отношению к ЭВМ является наличие в них нескольких вычислителей(процессоров или ЭВМ), реализующих параллельную обработку. Создание ВС преследует следующие основные цели: повышениепроизводительности системы за счет ускорения процессов обработки данных, повышение надежности и достоверности вычислений, предоставление пользователям дополнительных сервисных услуг и т.д.

В системах управления реального времени используются специализированные вычислительные системы, ориентированные нарешение узкого класса задач. Специализация ВС может обеспечиваться различными средствами:

- во-первых, сама структура системы (количество параллельно работающих элементов, связи между ними и т.д.) может бытьориентирована на определенные виды обработки информации: матричные вычисления, решение алгебраических, дифференциаль ных и интегральных уравнений и т.п. Практика разработки таких ВС показала, что, чем уже класс задач, для решения которых пред назначается специализированная ВС, тем большую производитель ность можно обеспечить;
- во-вторых, специализация ВС может закладываться включением в их состав специального оборудования и специальногопрограммного обеспечения.

Специализированные ВС по ***виду составляющих элементов*** принято подразделять на *многомашинные* и *многопроцессорные*.

Многомашинная ВС (ММС) содержит несколько ЭВМ, каждая из которых имеет свою оперативную память и работает под управлением своей операционной системы. Обмен между ЭВМ идет с помощью специальных дополнительных программных и аппаратурных средств. Многомашинные ВС относятся к классу *систем с распределенной памятью*.

Многопроцессорная (мультипроцессорная) система (МПС) в качестве общего ресурса имеет общую оперативную память (ООП).

Все процессоры работают с единым адресным пространством. Параллельная работа процессоров и использование ООП обеспечивается единой операционной системой. Все процессоры «разделяют» общую память, поэтому многопроцессорные ВС относятся к классу *систем с разделяемой памятью*.

По ***типам ЭВМ или процессоров***, из которых состоит ВС, раз личают однородные и неоднородные системы. *Однородные* систе мы предполагают комплексирование однотипных ЭВМ (процессо ров), *неоднородные* – разнотипных. В однородных системах значи тельно упрощается разработка и обслуживание технических и про граммных средств. В них обеспечивается возможность стандарти зации и унификации соединений и процедур взаимодействия эле ментов системы. Упрощается обслуживание систем, облегчается мо дернизация и их развитие. Вместе с тем существуют и неоднородные ВС, в которых комплексируемые элементы очень сильно отличаются по своим техническим и функциональным характеристикам и могут представлять собой специализированные процессоры.

По ***характеру пространственного распределения*** вычислительных модулей ВС делятся на системы *сосредоточенного (локального)* и *распределенного* типов. Обычно такое деление касается только ММС.

В локальных и распределенных ММС сильно различается опе ративность взаимодействия в зависимости от удаленности ЭВМ. Время передачи информации между соседними ЭВМ, соединенны ми простым кабелем, может быть много меньше времени передачи данных по каналам связи. Как правило, все выпускаемые в мире ЭВМ имеют средства прямого взаимодействия и средства подклю чения к сетям ЭВМ.

По ***методам управления элементами ВС*** различают *централизованные, децентрализованные* и *со смешанным управлением***.** Помимо параллельных вычислений, производимых элементами системы, необходимо выделять ресурсы на обеспечение управления этими вычислениями. В *централизованных ВС* за это отвечает главная, или диспетчерская, ЭВМ (процессор). Ее задачей является распределение нагрузки между элементами, выделение ресурсов, контроль состояния ресурсов, координация взаимодействия. Структура ВС может быть одноуровневой или иерархической. Централизованный орган управления в системе может быть жестко фиксирован, или эти функции могут передаваться другой ЭВМ (процессору), что способствует повышению надежности системы. Централизованные системы имеют более простые операционные системы (ОС). В *децентрализованных системах* функции управления распределены между ее элементами. Каждая ЭВМ (процессор) системы сохраняет известную автономию, а необходимое взаимодействие между элементами устанавливается по специальным наборам сигналов. С развитием ВС и, в частности, сетей ЭВМ интерес к децентрализованным системам постоянно растет.

В системах со *смешанным управлением* совмещаются процедуры централизованного и децентрализованного управления. Перераспределение функций осуществляется в ходе вычислительного процесса, исходя из сложившейся ситуации.

По ***принципу закрепления вычислительных функций*** за отдельными ЭВМ (процессорами) различают системы с *жестким и плавающим закреплением функций***.** В зависимости от типа ВС следует решать задачи статического или динамического размещенияпрограммных модулей и массивов данных, обеспечивая необходимую гибкость системы и надежность ее функционирования.

Многомашинные вычислительные системы

Исторически многомашинные вычислительные системы появились первыми. Уже при использовании ЭВМ первых поколений возникали задачи повышения производительности, надежности и достоверности вычислений.

Многомашинный комплекс состоит из нескольких ЭВМ, каждая из которых имеет свою внутреннюю память и работает под управлением своей операционной системы, и средства обмена информацией между машинами. В отличие от них мультипроцессорные ВС имеют общую память и общие периферийные устройства; они работают под управлением единой операционной системы.

Машины работают параллельно; каждая из них обрабатывают собственные потоки заданий, а в системах управления решает собственное подмножество функциональных задач. Несколько ЭВМ только тогда представляют собой вычислительную систему, когда они могут обмениваться информацией. Для этого требуется их совместимость.

Аппаратурная совместимость означает, что:

- подключаемые друг к другу элементы (узлы, блоки) должны иметь единые стандартные, унифицированные средства соединения: кабели, число проводов в них, единое назначение проводов, разъемы, заглушки, адаптеры, платы и т.д.;
- параметры электрических сигналов, которыми обмениваются элементы (узлы, устройства), тоже должны соответствовать друг другу: амплитуды импульсов, полярность, длительность и т.д.;
- должна быть решена проблема синхронизации вычислений;
- должна быть согласована форма представления чисел и их разрядность.

Программная совместимость предполагает, что:

- существует программа-диспетчер, которая оперативно распределяет ресурсы ВС между задачами, обеспечивает загрузку программ и управляет вводом-выводом;
- программы, которыми могут обмениваться ЭВМ, входящие в
- состав ВС, должны одинаково интерпретироваться ими (должнабыть согласована адресация, разрядности кодов команд коды операций и др.).

Эта проблема решается достаточно просто для однотипных ЭВМ.

Программная совместимость может обеспечиваться на уровнях:

1. внутреннего языка;
2. языка ассемблера;
3. языка высокого уровня.

На первом уровне наиболее распространены такие типы организации совместимости, как «общая наименьшая машина», иерархического типа, частичная и с эмуляцией недостающих команд.

Если на ЭВМ1 можно выполнять множество команд А, а на ЭВМ2 – множество команд В и пересечение этих множеств С не пусто, то оно соответствует возможностям «общей наименьшей машины».

Если одно множество полностью входит в другое, то имеет место иерархическая совместимость. Например, если взаимодействующие ЭВМ относятся к одному и тому же семейству, но стыкуются разные модели, то в таких моделях совместимость обеспечивается «снизу вверх», т.е. ранее созданные программы могут выполняться на более поздних моделях, но не наоборот.

Частичная совместимость возникает тогда, когда только часть программ, составленных для одной ЭВМ, может выполняться на другой. Эмуляция отсутствующих команд служит для увеличения степени частичной совместимости: если выполнение программы на малой ЭВМ прерывается из-за отсутствия в ней команд, существующих в большой ЭВМ, то эти команды могут эмулироваться программными средствами малой ЭВМ.

Совместимость на уровне машинных команд имеет следующие недостатки:

- разработка и отладка программ для обеспечения совместимости на этом уровне требует больших затрат времени и сил, особенно, для управляющих ЭВМ;
- эта совместимость приводит к неполному использованию специального оборудования и специальных команд, которые имеются не во всех ЭВМ;
- для надежного обмена программами требуется централизация их контроля и исправления ошибок.

Наиболее высокая степень программной совместимости – на третьем уровне – уровне алгоритмических языков, когда обмен программами, по существу, заменяется обменом алгоритмами. Совместимость на втором уровне имеет промежуточный характер.

Информационная совместимость предполагает, что передаваемые информационные массивы одинаково интерпретируются стыкуемыми модулями ВС. Должны быть стандартизированы алфавиты, разрядность, форматы, структура и разметка файлов, томов и т.д.

Эффективность ММС во многом зависит от организации обмена информацией и командами между ними. Характеристики обмена определяются параметрами технических средств, обеспечивающих взаимодействие устройств, и уровня управления процессами, на котором это взаимодействие осуществляется. В общем случае возможны следующие уровни комплексирования:

- прямого управления (процессор – процессор);
- общей оперативной памяти;
- комплексируемых каналов ввода-вывода;
- устройств управления внешними устройствами;
- общих внешних устройств.

Чаще всего комплексирование ЭВМ в ММС осуществляется на первом, втором и третьем уровнях.

Уровень прямого управления служит для передачи коротких приказов-сообщений. Последовательность взаимодействия процессоров сводится к следующему. Процессор ЭВМ-инициатора обмена передает в блок прямого управления сообщение и подает команду «Прямая запись». У процессора другой ЭВМ эта команда вызывает прерывание, относящееся к классу внешних. В ответ он вырабатывает команду «Прямое чтение» и записывает передаваемый байт в свою память. Затем принятая информация

расшифровывается и по ней принимается решение. После завершения передачи прерывания снимаются, и оба процессора продолжают вычисления по собственным программам. Очевидно, что уровень прямого управления не может использоваться для передачи больших массивов данных, однако оперативное взаимодействие отдельными сигналами широко используется в управлении вычислениями.

Уровень комплексируемых каналов ввода-вывода предназначается для передачи больших объемов информации между блоками оперативной памяти сопрягаемых ЭВМ. Обмен данными между ЭВМ осуществляется с помощью адаптера «канал-канал» и команд «Чтение» и «Запись». Адаптер – это устройство, согласующее скорости работы сопрягаемых каналов. Обычно сопрягаются селекторные каналы машин как наиболее быстродействующие, но можно сопрягать мультиплексные каналы, а также селекторный и мультиплексный. Скорость обмена данными определяется скоростью самого медленного канала. Скорость передачи данных по этому уровню составляет несколько Мбайт/с.

Уровень управления внешними устройствами предполагает использование встроенного в устройство управления ВУ (УВУ) двухканального коммутатора и команд «Зарезервировать» и «Освободить». Коммутатор позволяет подключать УВУ одной машины к селекторным каналам различных ЭВМ. По команде «Зарезервировать» канал-инициатор обмена получает доступ через УВУ к любым накопителям на дисках или на магнитных лентах. Обмен канала с накопителями продолжается до полного завершения работ и получения команды «Освободить». Лишь после этого УВУ может подключиться к конкурирующему каналу. Такая дисциплина обслуживания требований позволяет избежать конфликтных ситуаций.

Пять уровней комплексирования получили название *логических*, так как они объединяют на каждом уровне разнотипную аппаратуру, имеющую сходные методы управления. Каждое из устройств может иметь логическое имя, используемое в прикладных программах. Этим достигается независимость программ пользователей от конкретной физической конфигурации системы. Связь логической структуры программы и конкретной физической структуры ВС обеспечивается операционной системой по указаниямдирективам пользователя, при генерации ОС и по указаниям диспетчера-оператора вычислительного центра. Различные уровни комплексирования позволяют создавать самые различные структуры ВС.

Если в составе ВС имеется дорогое уникальное внешнее устройство, то возможно совместное его использование всеми машинами вычислительного комплекса – это еще один уровень комплексирования.

Многомашинные ВС применяются в системах управления сложными объектами. В таких системах с помощью ММС решают две задачи: обеспечение требуемой производительности и надежности. Применение ММС для обеспечения высоких требований по надежности функционирования и по достоверности информации обычно реализуется по следующей схеме.

Машины решают одну и ту же задачу, одновременно выполняя одинаковые операции над одними и теми же данными и периодически сверяя результаты. Если они совпадают, работа ММС продолжается. В случае несовпадения с помощью тестов или по результатам контроля определяется неисправная ЭВМ, ее выходы блокируются и работу системы обеспечивает исправная машина. После ремонта ММС вновь функционирует в нормальном режиме. Для этого в отремонтированную ЭВМ должна быть загружена обновленная информация и вновь обеспечен синхронный режим совместной работы.

Структурная организация ММС сильно зависит от конкретного применения, но можно указать основные концепции их построения:

- многоуровневая иерархическая организация структуры, соответствующая иерархическому характеру комплекса решаемых функциональных задач;
- динамическое перераспределение задач по ресурсам системы;
- модульность структуры и конструкции элементов ВС, обеспечиваемая унифицированной системой модулей и протоколов обмена;
- наличие развитых средств контроля и диагностики.

План:

- Поток данных и поток команд
- Слабосвязанные и сильносвязанные МКМД-системы
- Кластеры

Многопроцессорная система (МПС) – это совокупность процессорных элементов, которые взаимодействуют в процессе решения одной вычислительной задачи под управлением общей операционной системы. Известно много различных структур МПС и несколько принципов их классификации. Наибольшее распространение получила классификация М.Флинна (M.Flynn), построенная на концепции потока.

Если анализировать ВС с точки зрения сочетания потоков данных и потоков команд (понимая под *потоком команд* последовательность команд одной программы, а под *потоком данных* – последовательность данных, обрабатываемых одной программой), то возможны следующие варианты сочетаний:

1) одиночный поток команд – одиночный поток данных (ОКОД, или SISD – *Single Instuction Stream/ Single Data Stream*);
2) множественный поток команд – одиночный поток данных (МКОД, или MISD – *Multiple Instuction Stream/Single Data Stream*);
3) одиночный поток команд –множественный поток данных (ОКМД, или SIMD – *Single Instuction Stream/ Multiple Data Stream*);
4) множественный поток команд – множественный поток данных (МКМД, или MIMD – *Multiple Instuction Stream / Multiple Data Stream*).

Архитектура ОКОД (рис. 4.4.1, *а*) не представляет интереса для анализа как вычислительная система: это последовательная однопроцессорная скалярная или суперскалярная структура, возможно, с конвейерной обработкой. На одном процессоре выполняется один поток команд; операции выполняются над данными, которые хранятся в одной области памяти. К **архитектуре МКОД** (рис. 4.4.1, *б*) относят многопроцессорные конвейерные структуры.

Последовательность данных Д передается набору процессоров Пр1, Пр2,...,Пр*n*, каждый из которых выполняет свою последовательность команд. Примером может служить устройство для сложения двух векторов, компоненты которых – числа с плавающей точкой – последовательно поступают на вход Пр1. Процессоры Пр2, …, Пр*n* выполняют операции сравнения порядков, выравнивания, сложения мантисс, нормализации в режиме конвейера (параллельно) над всеми компонентами вектора. Следует заметить, что единого мнения по классу MISD нет: есть утверждения, что подобные структуры на практике еще не были реализованы; возможно, этот класс пуст.

Архитектура ОКМД – многопроцессорная матричная структура (рис. 4.4.1, *в*). Массив процессоров обрабатывает поток команд, поступающих от центрального управляющего устройства. Каждая команда выполняется всеми процессорами одновременно над разными потоками данных.

Примером применения матричной ВС может служить выполнение операций типа вычисления определенного интеграла:

$$\int_a^b y(x)dx \approx h[(y_0/2) + y_1 + y_2 + \cdots + y_{n-2} + (y_{n-1}/2)],$$

где $y_i = y(x_i), x_0 = a, x_n = b, h = b - a/n$. Каждый процессор выполняет одну и ту же операцию $hy(x_i)$.

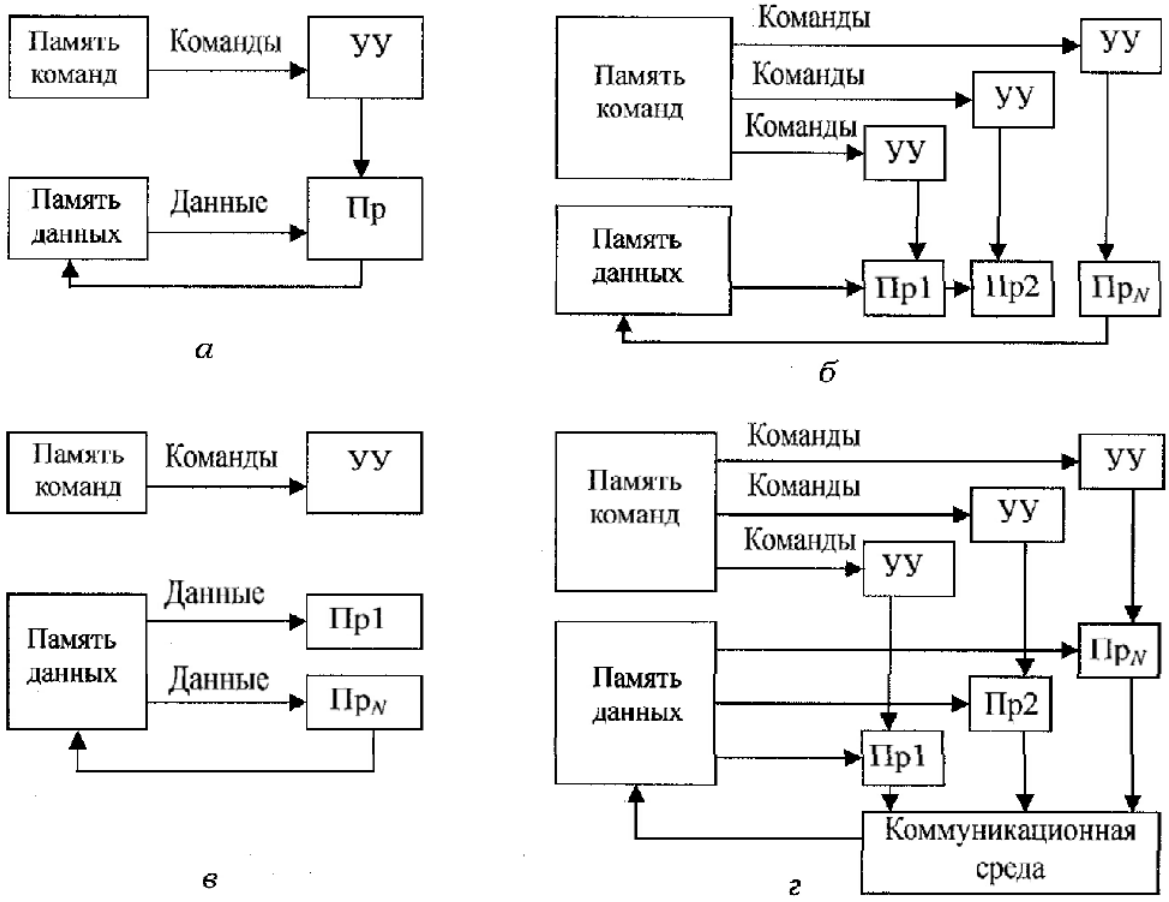

Рис. 4.4.1 - Многопроцессорная матричная структура

Главное отличие матричной ВС от векторного процессора заключается в том, что во втором случае высокая производительность обеспечивается с помощью интенсивной конвейеризации, а в первом – за счет высокой степени параллелизма.

Вычислительная система с **архитектурой типа МКМД** (рис. 4.4.1, *г*) выполняет одновременно и независимо друг от друга несколько программных ветвей с различными наборами данных, обеспечивая в определенные моменты времени обмен данными. В схеме МКМД процессоры должны быть универсальными, поскольку они должны выполнять все команды, необходимые для соответствующего преобразования данных. В классе МКМД различаются слабо- и сильносвязанные системы.

Слабосвязанные МКМД-системы – структуры с распределенной памятью. Десять-пятнадцать лет тому назад этот класс ВС состоял исключительно из многомашинных вычислительных комплексов и сетей ЭВМ. Вычислительные системы с такой структурой в качестве узлов используют законченные ЭВМ, включающие микропроцессор, память и подсистему ввода-вывода. Они объединяются коммуникационной средой, которая

обеспечивает взаимодействие процессоров посредством простых операций ввода/вывода (рис. 4.4.2, *а*). Каждый процессор Пр1,...,Пр*N* имеет свой модуль памяти и устройство ввода-вывода. Обмен идет через коммуникационную среду.

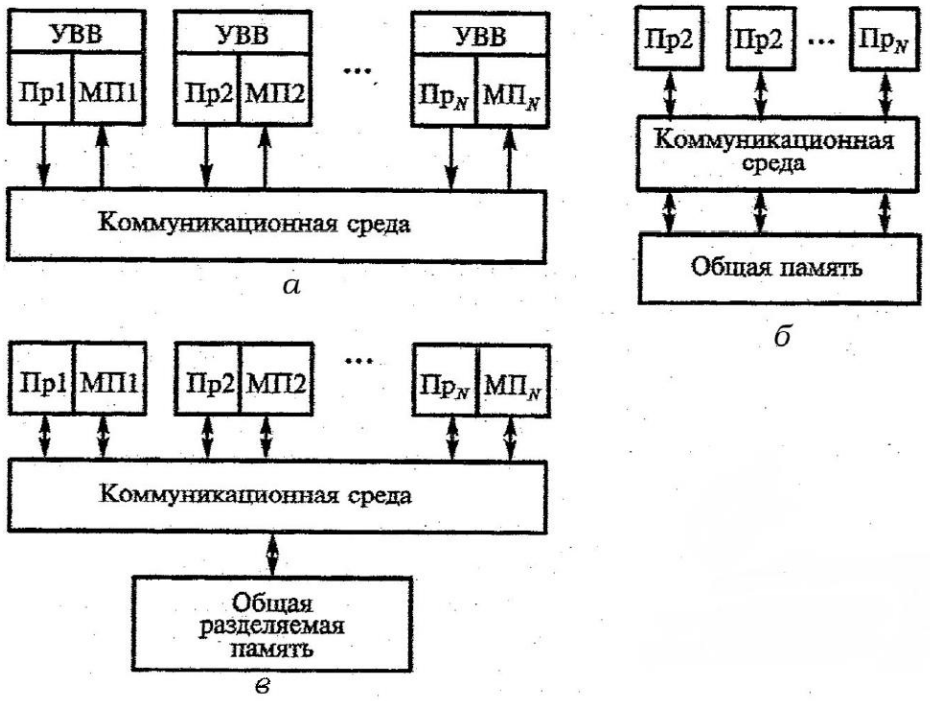

Рис. 4.4.2 - Взаимодействие процессоров с памятью

Сейчас в качестве элементов МКДМ-систем используются микропроцессоры. Пример современной слабосвязанной МКДМсистемы – Cray T3D/T3E. В нем память физически распределена, так как каждый процессор содержит свою локальную память. В то же время память разделяется всеми процессорами, поскольку каждый из них через свой сетевой интерфейс может обращаться к памяти любого другого процессора, не прерывая его работы. Коммуникации осуществляются с помощью средств операционной системы. Передача данных из одного локального адресного пространства к другому произойдет только в том случае, если посылка сообщения со стороны процесса-отправителя будет востребована процессом-получателем сообщения. В последнее время успехи микроэлектроники сделали возможным построение слабосвязанных систем, состоящих из сотен и тысяч процессорных элементов с возможностью наращивания их количества. Подобные структуры называют **системами с массовым параллелизмом** (MPP-системы – *Mass-Parallel Processing*). Увеличение количества процессорных элементов в ВС не может не привести к росту сложности коммуникационной системы, а также к новым идеям в области управления вычислительным процессом: от управления потоком команд можно перейти к управлению от потока данных, когда операторы программы активизируются по мере готовности данных.

У вычислительных систем с распределенной памятью соотношение цена/производительность ниже, чем у ВС других классов.

Для пользователя еще одно достоинство ВС этого класса заключается в возможности точно подобрать конфигурацию в зависимости от бюджета и своих требований к характеристикам вычислительных средств.

Сильносвязанные МКМД-системы – структуры с разделяемой общей памятью. Процессоры совместно используют общую память и каждый из них имеет равные права доступа к программам и данным, которые там хранятся; процессоры могут обмениваться информацией через единое адресное пространство памяти. Такие ВС называются *мультипроцессорными системами с общей памятью*.

Существует два варианта организации общей памяти:

1) все процессоры связываются с модулями общей памяти через коммуникационную среду; время обращения к памяти у всех процессоров одинаково и не зависит от того, к какому модулю памяти процессор обращается (см. рис. 4.4.2, *б*); такие МПС называют *системами с однородным доступом*;
2) каждый процессор имеет доступ к общей разделяемой памяти, но, кроме того, он может обращаться и к собственной локальной памяти (см. рис. 4.4.2, *в*); системы этого типа называют *системами с неоднородным доступом.* Доступ к локальной памяти идет намного быстрее, чем к общей, что повышает производительность ВС. В классе сильносвязанных МКМД-систем выделяют два типа архитектуры: асимметричную и симметричную.

В асимметричной архитектуре разные процессоры могут отличаться как своими характеристиками (производительностью, надежностью, системой команд и т.д.), так и функциональной ролью, которая поручается им в системе. Например, одни процессоры могут предназначаться для работы в качестве вычислительных модулей, другие – для управления вводом/выводом.

Симметричная архитектура предполагает однородность всех процессоров и единообразие их включения в общую схему МВС.

В качестве проблем, которые характерны для параллельных ВС с общей памятью указываются:

- высокая стоимость;
- неоднородность доступа к памяти: в то время как один процессор обращается к локальной памяти, другой может обращаться к удаленной – время доступа различно и это должно учитываться при организации общего вычислительного процесса;
- необходимость обеспечения согласованности содержимого кэш-памяти.

Сравнение многопроцессорных и многомашинных вычислительных систем приводит к следующим выводам. Многопроцессорные ВС обеспечивают более быстрый обмен между процессорами, более высокую надежность и живучесть по сравнению с ММС.

Однако МПС имеют и существенные недостатки. Они, в первую очередь, связаны с использованием ресурсов общей оперативной памяти. При большом количестве комплексируемых процессоров возможно возникновение конфликтных ситуаций, когда несколько процессоров одновременно обращаются с операциями типа «чтение» и «запись» к одним и тем же областям памяти. Помимо процессоров к общей памяти, подключаются все каналы (процессоры ввода/вывода), средства измерения времени и т.д. Поэтому вторым серьезным недостатком МПС является проблема коммутации абонентов и доступа их к общей памяти.

Создание коммутаторов оперативной памяти представляет сложную техническую задачу: для разрешения конфликтных ситуаций необходимы схемы приоритетного обслуживания; они должны быть дополнены буферами для организации очередей запросов. До настоящего времени в номенклатуре технических средств вычислительной техники отсутствуют высокоэффективные коммутаторы общей памяти.

От того, насколько удачно решаются эти проблемы, и зависит эффективность применения МПС. Это решение обеспечивается аппаратно-программными средствами. Процедуры взаимодействия процессоров очень сильно усложняют структуру операционной системы МПС. Накопленный опыт построения подобных систем показал, что они эффективны при небольшом числе комплексируемых процессоров (от 2 до 10). В отечественных системах «Эльбрус» обеспечивалась возможность работы до 10 процессоров, до 32 модулей оперативной памяти, до четырех процессоров ввода-вывода и до восьми коммутаторов оперативной памяти.

Кластеры

Кластером называют группу взаимосвязанных совместно работающих ЭВМ, представляющих для пользователя единый ресурс, единый центр по обработке информации. В качестве вычислительного узла обычно используются двух- или четырехпроцессорные ЭВМ, хотя могут устанавливаться и однопроцессорные ЭВМ, и локальные сети. Каждый узел работает под управлением своей операционной системы. Состав и мощность узлов могут меняться даже в рамках одного кластера, что дает возможность создавать неоднородные системы. Возможно включение в конфигурацию кластера специализированного компьютера, например, файл-сервера. Поскольку все ЭВМ кластера могут работать самостоятельно и в качестве узлов кластера могут выступать вычислительные системы, то кластер фактически представляет собой вычислительную систему более высокого уровня иерархии.

Концепция кластера появилась как следствие двух факторов:

1) стремления повысить производительность вычислительного ресурса, представляемого пользователю.
2) потребностью в высоконадежных средствах обработки информации.

Достоинства кластерных структур определяются следующим образом.

1. Можно создавать кластеры любых размеров, превышающие по суммарной производительности любые самые мощные ЭВМ. Кластер может состоять из десятков и сотен ЭВМ, каждая и которых может быть многопроцессорной вычислительной системой. Это свойство наращивать мощность (*масштабируемость* кластера) обеспечивается аппаратурной, программной и информационной совместимостью его элементов и автоматизацией процесса распределения нагрузки.
2. Пользователь может наращивать технические средства постепенно, и при этом вычислительная мощность кластера будет возрастать линейно, пропорционально приращению характеристик технических средств. Для других типов ВС подобная зависимость имеет нелинейный характер вследствие роста числа конфликтов и усложнения системы обмена.

Возможны два решения проблемы масштабируемости. *Вертикальное* масштабирование осуществляется добавлением ЭВМ, модулей памяти, процессоров и

средств обмена в каждый узел кластера. *Горизонтальное* масштабирование состоит в увеличении числа узлов и в добавлении внешней памяти.

3. Кластер обладает повышенной надежностью и готовностью. Это свойство обеспечивается способностью каждого узла кластера работать автономно, но в любой момент переключаться на выполнение работ другого узла в случае его отказа.
4. Поскольку в качестве «строительных блоков» кластера используются широко распространенные ЭВМ, его стоимость, как правило, меньше стоимости одиночной уникальной ЭВМ с характеристиками, аналогичными обобщенным характеристикам кластера.

Простейший способ обеспечить свойства масштабируемости, надежности и готовности – построить кластер, состоящий из двух узлов, соединенных высокоскоростной линией связи. Эта связь может представлять собой локальную вычислительную сеть, которая используется либо совместно с другими ЭВМ, либо специально для нужд кластера. Дальнейшее развитие этой схемы заключается в дополнительной связи между ЭВМ через дисковую подсистему. Использование избыточной внешней памяти типа RAID (*Redundant Array of Inexpensive Disk* – избыточный массив недорогих дисков) типично для кластеров, оно обеспечивает их высокую надежность.

Для RAID-массивов было разработано шесть разных конфигураций, отвечающих различным задачам. Они названы уровнями RAID, хотя никакой иерархии не представляют.

RAID 0 – это базовая конфигурация дискового массива, предназначенная для повышения производительности системы. Один большой файл разбивается на несколько частей, которые записываются на разные диски. Это называется *расслоением данных*. При обращении к такому файлу для чтения диски могут передавать данные параллельно, так что общее время его пересылки сокращается по сравнению со временем пересылки при хранении на одном диске.

Архитектура RAID 1 позволяет повысить надежность хранения данных путем записи их идентичных копий на двух дисках. Такие диски называют *зеркальными*. Если один из них выходит из строя, операции чтения и записи продолжаются с зеркальным диском.

Уровни RAID 2, RAID 3 и RAID 4 предназначены для повышения надежности системы с помощью различных схем контроля четности, не требующих полного дублирования дисков. Вся информация, предназначенная для контроля четности, хранится на диске. В RAID 5 она распределяется между всеми дисками.

Архитектура кластеров используется при построении серверов для распределенных вычислительных систем.

В распределенных ВС появляются распределенные базы данных. Распределенная база данных (РБД) – это база, отдельные части которой размещены (возможно, с дублированием) на нескольких ЭВМ, объединенных сетью. Она должна удовлетворять следующим требованиям:

- быстрая обработка типовых и незапланированных запросов;
- обеспечение безопасности данных;
- обеспечение логической и физической независимости данных;
- обеспечение «прозрачности»:
  - распределенной структуры данных (независимость конечных пользователей и программ от варианта размещения информации на ЭВМ сети);

- совместного доступа к данным (поддержка целостности РБД при одновременной модификации одних и тех же данных несколькими пользователями);
- распределенной обработки (независимость конечных пользователей и прикладных программ от типа сети и применяемой сетевой операционной системы).

Администрирование и доступ пользователей к РБД осуществляются с помощью системы управления РБД, которая, исходя из упомянутых требований, обеспечивает выполнение следующих функций:

- автоматическое определение ЭВМ, на которой хранятся требуемые в запросе данные;
- декомпозицию распределенных запросов на частные подзапросы к БД отдельных ЭВМ;
- планирование обработки запросов;
- передачу частных подзапросов и их исполнение на удаленных ЭВМ;
- прием результатов выполнения частных подзапросов, полученных в результате декомпозиции запросов на поиск и чтение данных из РБД, и композицию общего результата;
- поддержание в согласованном состоянии копий дублированных данных на различных ЭВМ сети;
- управление параллельным доступом к РБД многих пользователей;
- обеспечение целостности РБД.

Реализация перечисленных выше функций в распределенной ВС поручается **серверам**. Вообще под термином «сервер» принято понимать объект, предоставляющий сервис другим объектам по их запросам; под *сервисом* понимается процесс обслуживания объектов, а наиболее распространенными видами сервиса являются такие как: хранение и поиск данных; передача сообщений и блоков данных; электронная почта; организация и управление диалогом пользователей; предоставление соединений; проведение сеансов взаимодействия прикладных процессов.

Существует множество типов серверов:

- сервер баз данных, выполняющий обработку запросов, направляемых базе данных;
- файловый сервер, обеспечивающий функционирование распределенных ресурсов, включая файлы и программное обеспечение;
- сервер доступа, обеспечивающий возможность клиенту одной локальной сети связываться с другой сетью;
- коммуникационный сервер, управляющий терминалами и осуществляющий маршрутизацию сообщений.

Распределенные ВС можно строить по принципам, получившим названия «файл – сервер» и «клиент – сервер».

В ВС, построенной по принципу «**файл–сервер»**, файловый сервер не принимает участия в реализации прикладной программы пользователя. Он просто хранит файлы, хотя в ряде случаев служит концентратором для совместного использования периферийных устройств, организуя очереди к периферийным устройствам. Так как файловый сервер является независимым компьютером сети, то его обычно специализируют, устанавливая, например, большой объем дисковой памяти. Функционирующая на рабочей станции прикладная программа считывает и записывает файлы, обмениваясь ими с сетевым файловым сервером, но все операции выполняет самостоятельно.

Во многих случаях файлы передаются по сети между рабочей станцией и сервером целиком, хотя пользователю нужна только их часть. Это приводит к неэффективной загрузке сети. Другой недостаток систем типа «файл – сервер»: в них невозможен одновременный доступ к одному набору данных многими пользователями.

Присущие локальной сети проблемы породили идею кооперативных вычислений по принципу «**клиент–сервер»**. *Клиент* в такой системе – это процесс, который с помощью определенных операций вызывает сервисную функцию. Им может быть программа или пользователь. Сервисная функция в архитектуре «клиент–сервер» описывается комплексом прикладных программ, в соответствии с которым выполняются прикладные процессы. Функции сервера реализуются в виде самостоятельного прикладного процесса либо части общего прикладного процесса. После выполнения каждого задания сервер посылает результаты клиенту, пославшему это задание.

Функции сети связи известны. В этой системе нагрузка на нее значительно ниже, чем в системе с файловым сервером, поскольку обмен идет не целыми массивами данных, а отдельными фрагментами, например, строками таблицы.

Достоинства системы организации вычислений типа «клиент–сервер» определяются, во-первых, тем, что клиентская и серверная часть могут работать на разных по стоимости и по производительности машинах, что приводит к снижению затрат на систему. Вовторых, такие системы обладают хорошей адаптацией и гибкостью: замена сервера не приводит к нарушению клиентских приложений, а расширение числа пользователей (в определенных пределах) не влияет на работу сервера. В-третьих, каждый функциональный элемент, в том числе и клиентское приложение, легко может быть специализировано для выполнения тех или иных специфических функций.

По общему признанию, вычисления по схеме «клиент–сервер» более дешевые по сравнению с централизованными системами с большими ЭВМ.

В то же время нельзя забывать о том, что отказы сервера делают всю систему неработоспособной, а применение этой схемы для системы, обрабатывающей сотни и тысячи одинаковых приложений (типа заказов билетов), сводят на нет все ее достоинства.

ГЛАВА-IV. ПРОЦЕССОРЫ

Лекция 4.1. Архитектура процессоров.

План:

- CISC и RISC процессоры
- Конвейризация
- Основные черты RISC архитектуры
- Кэш память
- Поддержка многозадачности и многопроцессорности

CISC и RISC процессоры

Основной временной характеристикой для процессора является время цикла, равное 1/**F**, где **F** - тактовая частота процессора. Время, затрачиваемое процессором на задачу, может быть вычислено по формуле **C** * **T** * **I**, где **C** - число циклов на одну инструкцию, **T** - время на один цикл, **I** - число инструкций на задачу.

Разработчики “классических” систем (которые теперь называют CISC (Complete Instruction Set Computer)) стремились уменьшить фактор **I**. В процессорах реализовывались все более сложные инструкции, для выполнения которых внутри него самого запускались специальные процедуры (так называемый микрокод), загружаемые из ПЗУ внутри процессора. Этому пути способствовало то, что улучшения в технике производства полупроводников делали возможным реализацию все более сложных интегрированных цепей. Однако, на этом пути очень трудно уменьшить два других фактора: **C** поскольку инструкции сложные и требуют *программного декодирования* и **T** в силу *аппаратной сложности.*

Концепция RISC (Reduced Instruction Set Computer) возникла из статистического анализа того, как программное обеспечение использует ресурсы процессора. Исследования системных ядер и объектных модулей, порожденных оптимизирующими компиляторами, показали подавляющее доминирование простейших инструкций даже в коде для CISC машин. Сложные инструкции используются редко, поскольку микрокод обычно не содержит в точности те процедуры, которые нужны для поддержки различных языков высокого уровня и сред исполнения программ. Поэтому разработчики RISC процессоров убрали реализованные в микрокоде процедуры и передали программному обеспечению низкоуровневое управление машиной. Это позволило заменить процессорный микрокод в ПЗУ на подпрограмму в более быстрой ОЗУ.

Разработчики RISC процессоров улучшили производительность за счет уменьшения двух факторов: **C** (за использования только простых инструкций) и **T** (за счет упрощения процессора). Однако, изменения, внесенные для уменьшения числа циклов на инструкцию и времени на цикл, имеют тенденцию к увеличению числа инструкций на задачу. Этот момент был в центре внимания критиков RISC архитектуры. Однако, использование *оптимизирующих компиляторов* и других технических приемов, практически ликвидирует эту проблему.

Основные черты RISC архитектуры

У RISC процессора все инструкции имеют одинаковый формат и состоят из битовых полей, определяющих код инструкции и идентифицирующих ее операнды. В силу этого *декодирование инструкций производится аппаратно,* т.е. микрокод не требуется. При этом в силу одинакового строения всех инструкций процессор может декодировать несколько полей *одновременно* для ускорения этого процесса.

Инструкции, производящие операции в памяти, обычно либо увеличивают время цикла, либо число циклов на инструкцию. Такие инструкции требуют дополнительного времени для своего исполнения, так как требуется вычислить адреса операндов, считать их из памяти, вычислить результат операции и записать его обратно в память. Для уменьшения негативного влияния таких инструкций, разработчики RISC процессоров выбрали *архитектуру чтение/запись,* в которой все операции выполняются над операндами в регистрах процессора, а основная память доступна только посредством инструкций чтения/'записи. Для эффективности этого подхода RISC процессоры имеют *большое количество регистров.* Архитектура чтение/'запись также позволяет *уменьшить количество режимов адресации памяти,* что позволяет упростить декодирование инструкций.

Для CISC архитектур время исполнения инструкции обычно измеряется в числе циклов на инструкцию. Разработчики RISC архитектур, однако, стремились получить скорость выполнения инструкции, равную *одной инструкции за цикл.*

Для RISC процессора во многих случаях только наличие *оптимизирующего компилятора* позволяет реализовать все его возможности. Отметим, что компилятор может наилучшим образом оптимизировать код именно для RISC архитектур (в силу их простоты). Программирование на языке ассемблера исчезает для RISC приложений, так как компиляторы языков высокого уровня могут производить очень сильную оптимизацию.

Конвейеризация

Конвейеризация является одним из основных способов повышения производительности процессора. Конвейерный процессор принимает новую инструкцию каждый цикл даже если предыдущие инструкции не завершены. В результате выполнение нескольких инструкций перекрывается и в процессоре находятся сразу несколько инструкций в разной степени готовности.

Исполнение инструкций может быть разделено на несколько стадий:

- выборка,
- декодирование,
- исполнение,
- запись результатов.

Конвейер инструкций может уменьшить число циклов на инструкцию посредством одновременного исполнения нескольких инструкций, находящихся на разных стадиях. При правильной аппаратной реализации конвейер, **имеющий** *n* стадий может одновременно исполнять *n* последовательных **инструкций.** Новая инструкция может приниматься к исполнению на каждом цикле, и эффективная скорость исполнения, таким образом, есть

один цикл на инструкцию. Однако, это предполагает, что конвейер всегда заполнен полезными инструкциями и нет задержек в прохождении инструкций через конвейер.

Управление конвейером инструкций требует надлежащего эффективного управления такими событиями, как **переходы, исключения** или **прерывания,** которые могут полностью нарушить поток инструкций. Например, результат условного перехода известен, только когда эта инструкция будет исполнена. Если конвейер был заполнен инструкциями, следующими за инструкцией условного перехода и переход состоялся, то все эти инструкции должны быть выброшены из конвейера.

Более того, внутри конвейера могут оказаться **взаимозависимые** инструкции. Например, если инструкция в стадии декодирования должна читать из ячейки памяти, значение которой является результатом работы инструкции, находящейся в стадии исполнения, то конвейер будет остановлен на один цикл, поскольку этот результат будет доступен только после стадии записи результатов. Поэтому компилятору необходимо **переупорядочить** инструкции в программе так, чтобы по-возможности избежать зависимостей между инструкциями внутри конвейера.

Для уменьшения времени простоя конвейера применяют ряд мер.

- **Таблица регистров (register scoreboarding)** позволяет проследить за использованием регистров. Она имеет бит для каждого регистра процессора. Если этот бит установлен, то регистр находится в состоянии ожидания записи результата. После записи результата этот бит сбрасывается, разрешая использование этого регистра. Если этот бит сброшен для всех регистров, значения которых используются в текущей инструкции, то ее можно выполнять, не дожидаясь завершения исполнения предыдущих инструкций.
- **Переименование регистров (register renaming)** является аппаратной техникой уменьшения конфликтов из-за регистровых ресурсов. Компиляторы преобразуют языки высокого уровня в ассемблерный код, назначая регистрам те или иные значения. В суперскалярном процессоре операция может потребовать регистр до того, как предыдущая инструкция закончила использование этого регистра. Это состояние *не является конфликтом данных,* поскольку этой операции не требуется значение регистра, а только сам регистр. Однако, эта ситуация приводит к остановке конвейера до освобождения регистра. Идея разрешения этой проблемы состоит в следующем: берем свободный регистр, переименовываем его для соответствия параметрам инструкции, и даем инструкции его использовать в качестве требуемого ей регистра.

Задержки внутри конвейера могут быть также вызваны временем доступа к оперативной памяти DRAM, которое намного превышает время цикла. Эта проблема в значительной степени снимается при использовании кэш памяти и буфера предвыборки инструкций **(очереди инструкций).**

Так как поток инструкций в CISC процессоре **нерегулярный** и время исполнения одной инструкции **(C * T)** не постоянно, то конвейеризация в этом случае имеет серьезный недостаток, делающий ее малопригодной к использованию в CISC процессорах: именно, она приводит к очень сильному усложнению процессора.

RISC процессоры используют один и тот же формат всех инструкций для того, чтобы ускорить декодирование и упростить управление конвейером, поэтому все инструкции исполняются за **один** цикл.

Одним из способов дальнейшего повышения быстродействия является конвейеризация стадий конвейера. Такие процессоры называются **суперконвейерными.** При таком подходе каждая стадия конвейера, такая как кэш (см. ниже) или АЛУ (арифметическое и логическое устройство), может принимать новую инструкцию каждый цикл, даже если эта стадия не завершила исполнение текущей инструкции. Отметим, что добавление новых уровней конвейеризации имеет смысл только в случае, если разработчик может **значительно увеличить частоту** процессора. Однако, увеличение производительности за счет увеличения внутренней частоты процессора имеет ряд недостатков. Во-первых, это увеличивает потребление энергии процессором, что делает суперконвейерные процессоры малопригодными для встраиваемых систем. Во-вторых, это вводит новые трудности в сопряжении процессора с памятью нижнего уровня, такой как DRAM. Быстродействие этой памяти растет не так быстро, как скорость процессоров, поэтому чем быстрее процессор, тем больше разрыв в производительности между ним и основной памятью.

Другим способом увеличения производительности процессоров является выполнение более чем одной операции одновременно. Такие процессоры называются **суперскалярными.** Они имеют **два или более конвейеров инструкций,** работающих параллельно, что значительно увеличивает скорость обработки потока инструкций. Одним из достоинств суперскалярной архитектуры является возможность увеличения производительности без необходимости увеличения частоты процессора. Суперскалярному процессору требуется более **широкий** доступ к памяти, так, чтобы он мог брать сразу группу из нескольких инструкций для исполнения. **Диспетчер** анализирует эти группы и заполняет каждый из конвейеров так, чтобы снизить взаимозависимость данных и конфликты регистров. Выполнение инструкций может быть не по порядку поступления, так, чтобы команды перехода были проанализированы раньше, убирая задержки в случае осуществления перехода. Компилятор должен оптимизировать код для обеспечения заполнения всех конвейеров.

Требование одного и того же ресурса несколькими инструкциями блокирует их продвижение по конвейеру и приводит к вставке циклов ожидания требуемого ресурса. Суперскалярная архитектура с тремя исполняющими устройствами будет полностью эффективной, только если поток инструкций обеспечивает одновременное использование этих трех устройств. Для обеспечения этого условия с минимальными расходами в процессоре выделяют исполняющие устройства, работающие независимо:

- **Целочисленное устройство** (IU, Integer Unit) - выполняет целочисленные операции (арифметические, логические и операции сравнения) в своем АЛУ.
- **Устройство для работы с плавающей точкой** (FPU, Floating Point Unit) - обычно отделено от целочисленного устройства, которое работает только с целыми числами и числами с фиксированной точкой. Большинство FPU совместимы со стандартом ANSI/IEEE для двоичной арифметики с плавающей точкой. •
- **Устройство управления памятью** (MMU, Memory Management Unit) - вычисляет реальный физический адрес по виртуальному адресу.
- **Устройство предсказания переходов** (BU, Branch Unit) - занимается предсказанием условных переходов, для того, чтобы избежать простоя конвейера в ожидании результата вычисления условия перехода.

Поскольку условные переходы могут свести на нет все преимущества конвейерной организации процессора, остановимся более подробно на приемах, используемых BU для уменьшения их негативного влияния.

- **“Отложенные слоты”** (delay slots). Инструкцию, передающую управление от одной части программы другой, трудно исполнить за один цикл. Обычно загрузка процессорного указателя на следующую инструкцию требует один цикл, предвыборка новой инструкции требует еще один. Для избежания простоя, некоторые RISC процессоры (например, SPARC) позволяют вставить дополнительную инструкцию в так называемый “отложенный слот”. Эта инструкция, которая расположена непосредственно после команды перехода, но будет выполнена до того, как будет совершен переход.

  Однако, для суперскалярных RISC процессоров отложенные слоты работают не очень хорошо. Задержка при переходе может быть два цикла, а суперскалярный процессор, который выполняет за цикл *n* инструкций, должен найти *n* инструкций для помещения в конвейеры.

- **“Спекулятивное” исполнение инструкций** (speculative execution). Некоторые RISC процессоры (например, старшие модели семейств PowerPC и SPARC) используют так называемое ’спекулятивное’ исполнение инструкций: процессор загружает в конвейер и начинает исполнять инструкции, находящиеся за точкой ветвления, еще не зная, произойдет переход или нет. При этом часто выбирается наиболее вероятная ветвь программы (на основе того или иного подхода, см. ниже). Если после исполнения команды перехода оказалось, что процессор начал исполнять не ту ветвь, то все загруженные в конвейер инструкции из этой ветви и результаты их обработки сбрасываются, и загружается правильная ветвь.

- **Виты предсказания перехода в инструкции.** Некоторые RISC процессоры (например, PowerPC) используют биты предсказания перехода, которые устанавливает компилятор в инструкции перехода, и предсказывающие, будет или нет совершен переход.

- **Эвристическое предсказание переходов.** Некоторые RISC процессоры уменьшают задержки, вносимые переходами, за счет использования **встроенного предсказателя переходов.** Он предсказывает, что переходы вперед (проверки) произведены не будут, а переходы назад (циклы) - будут.

Для эффективной работы устройства предсказания перехода важно, чтобы код условия для условного перехода был вычислен как можно раньше (за несколько инструкций до самой команды перехода). Этого добиваются несколькими способами. •

- **Независимость арифметических операций и кода условия.** В CISC архитектурах все арифметические операции выставляют код условия по своему результату. Это сделано для уменьшения фактора I - числа инструкций на задачу, поскольку есть вероятность того, что следующая инструкция будет вычислять код условия по результату предыдущей инструкции и, следовательно, может быть удалена. Однако это приводит к тому, что между командой вычисления кода условия и командой перехода очень трудно вставить полезные инструкции, так как они изменят код условия. В RISC архитектурах арифметические операции *не изменяют код условия* (если противное явно не указано в инструкции, см. ниже). Поэтому возможно между инструкцией, вычисляющей код условия, и командой перехода вставить другие

инструкции (переупорядочив их). Это позволит заранее узнать, произойдет или нет переход и загрузить конвейер инструкциями.

Поскольку возможна ситуация, когда следующая инструкция будет вычислять код условия по результату предыдущей инструкции, то в RISC архитектурах часть (SPARC) или все (PowerPC) арифметические операции также имеют вторую форму, в которой будет выставляться код условия по их результату. Таким образом, часть или все арифметические операции присутствуют в двух вариантах: один не изменяет код условия (подавляющее большинство случаев использования), а другой вычисляет код условия по результату операции.

- **Использование нескольких равноправных регистров с кодом условия.** Некоторые RISC процессоры (например, PowerPC) используют несколько равноправных регистров, в которых образуется результат вычисления условия. Над этими регистрами определены логические операции, что иногда позволяет оптимизирующему компилятору заменить команды перехода при вычислении сложных логических выражений на команды логических операций с этими регистрами.
- **Использование кода условия в каждой инструкции.** Некоторые RISC процессоры (например, ARM) используют код условия в каждой инструкции. В формате каждой инструкции предусмотрено поле, где компилятором записывается код условия, при котором она будет выполнена. Если в момент исполнения инструкции код условия не такой, как в инструкции, то она игнорируется. Это позволяет вообще обойтись без команд перехода при вычислении результатов условных операций.

Кэш память

Время, необходимое для выборки инструкций, в основном зависит от подсистемы памяти и часто является ограничивающим фактором для RISC процессоров в силу высокой скорости исполнения инструкций. Например, если процессор может брать инструкции только из DRAM с временем доступа 60 ns, то скорость их обработки (при расчете одна инструкция за цикл) будет соответствовать тактовой частоте 16.7 MHz. Эта проблема в значительной степени снимается за счет использования **кэш памяти.**

Кэш память (cache) - это быстрое статическое ОЗУ (SRAM), вставленная между исполнительными устройствами и системным ОЗУ (RAM). Она сохраняет последние использованные инструкции и данные, так, что циклы и операции с массивами будут выполняться быстрее. Когда исполняющему устройству нужны данные и они не находятся в кэш памяти, то это **кэш-промах:** процессор должен обратиться к внешней памяти для выборки данных. Если требуемые данные находятся в кэше, то это **кэш-попадание:** доступ к внешней памяти не требуется.

Таким образом, кэши разгружают внешние шины, уменьшая потребность в них процессора. Это позволяет нескольким процессорам разделять внешние шины без уменьшения производительности каждого из них.

Кэш содержит строки из нескольких последовательных байтов (обычно 32 байта), которые загружаются процессором, используя так называемый *импульсный* (или блочный) *доступ* (burst access). Даже если CPU нужен один байт, все равно будет загружена целая строка, так как вероятно, что тем самым будут загружены следующие выполняемые инструкции или используемые данные. Блочные передачи обеспечивает высокие скорости

передачи для инструкций или данных в последовательных адресах памяти. При таких передачах только адрес первой инструкции или данного будет послан в подсистему внешней памяти. Все последующие запросы инструкций или данных в последовательных адресах памяти не требуют дополнительной передачи адреса. Например, загрузка 16 байтов требует 5 циклов, если MC68040 делает блочную передачу для загрузки строки кэша, и 8 циклов, если память не поддерживает блочный режим передачи.

Кэш, в котором вместе хранятся данные и инструкции, называется **единым кэшем.** Одним из способов повышения производительности является введение в процессоре *трех* шин: *адреса, инструкций* и *данных.* В **Гарвардской архитектуре кэша** разделяют кэши для инструкций и данных для удвоения эффективности кэш памяти. В типичной Гарвардской архитектуре присутствуют три вида кэш памяти: специальные кэши (например, TLB), внутренние кэши инструкций и данных **(первого уровня** или L1 кэш) и внешний единый кэш **(второго уровня** или L2 кэш). В процессорах, имеющих интегрированные кеши первого и второго уровней (т.е. внутри корпуса процессора), часто дополнительно устанавливают единый кэш **третьего уровня** (L3) вне процессора. Обычно L1 кэш работает на частоте процессора и имеет размер 8... 32Кб, L2 кэш работает на частоте процессора или ее половине и имеет размер 128Кб... 4Мб, L3 кэш работает на частоте внешней шины и имеет размер 512Кб... 8Мб.

Кэши данных в зависимости от их поведения при записи данных в кэш разделяют на два вида.

1. Кэш с прямой записью (write-through cache). Этот вид кэш памяти при записи в нее сразу инициирует цикл записи во внешнюю память. Основным достоинством такого кэша является простота и то, что данные в кэше и в памяти всегда идентичны, что упрощает построение многопроцессорных систем.
2. Кэш с обратной записью (write-back cache) Этот вид кэш памяти при записи в нее не записывает данные сразу во внешнюю память. Запись в память осуществляется при выходе строки из кэша или по запросу системы синхронизации в многопроцессорных системах. Такая организация кэш памяти может значительно ускорить выполнение циклов, в которых обновляется одна и та же ячейка памяти (будет записано только последнее, а не все промежуточные значения как в кэше с прямой записью). Другим достоинством является уменьшение потребности процессора во внешней шине, что позволяет разделять ее нескольким процессорам. Недостатком такой организации является усложнение схемы синхронизации кэшей в многопроцессорных системах.

В силу его значительно большей эффективности, большинство современных процессоров используют кэш с обратной записью.

Организация кэш-памяти. Кэш основан на **сравнении адреса.** Для каждой строки кэша хранится адрес ее первого элемента, называемый адресом строки. Для уменьшения объема дополнительно хранимой информации (адресов строк) и ускорения поиска адреса используют несколько технических приемов.

- Пусть длина строки есть 2^b байт. Адреса строк выровнены на границу своего размера, т.е. последние *b* бит адреса - нулевые и потому не хранятся (т.е. размер адреса уменьшен до 32-*b* бит).
- Фиксируется некоторое *i.* Строки хранятся как один или несколько *(N)* массивов, отсортированными по порядку *i* младших битов адреса (т.е. младших среди

оставшихся 32-*b*). Таким образом, *k*-й элемент массива имеет адрес, биты которого в позициях от 32-*i*-*b*+1 до 32-b образуют число, равное *к*. Это позволяет не хранить эти биты (т.е. размер адреса уменьшен до 32-*b*-*i* бит).

- Комбинация чисел *b* и *i* подбирается так, чтобы младших *b* + *i* бит логического адреса совпадали бы с соответствующими битами физического адреса при страничном преобразовании (т.е. были бы смещением в странице). Это позволяет параллельно производить трансляцию адреса и поиск в кэше (т.е. параллельно работать MMU и кэшу). При типичном размере страницы 4КЬ это означает *b*+*i*= 12.
- Определение того, содержится ли данный адрес в кэше, производится следующим образом. Берутся биты в позициях от 32-*i*-*b*+1 до 32-*b*, образующие число *k*. Затем берутся элементы с номером *k* в каждом из *N* массивов и *y* полученных *N* строк сравниваются адреса с 32-*i*-*b* битами адреса (которые уже транслированы MMU в физический адрес). Если обнаружено совпадение (т.е. имеет место кэш-попадание), то берется байт с номером *b* в строке.

Если рассматривается внешний кэш, то согласовывать его работу с MMU не требуется, поскольку внешний кэш работает уже с физическим адресом.

Пример: для PowerPC 603 выбрано b = 5 (т.е. длина строки 32 байта), *i*=7 (т.е. длина массива 127), *N* = 2 (т.е. используются два массива).

Для каждой строки кэша с обратной записью помимо адреса хранится также признак того, что эта строка содержит корректные данные, т.е. данные в кэш памяти и в основной памяти совпадают. Этот признак используется для записи строки в память при ее выходе из кэш памяти, а также в многопроцессорных системах.

Алгоритмы замены данных в кэш памяти. Если все строки кэш памяти содержат корректные данные, то для обеспечения кэширования новых областей памяти необходимо выбрать строку, которая будет перезаписана. Эта строка выходит из кэша и, если требуется, ее содержимое будет записано обратно в память. Существуют три алгоритма замены данных в кэше:

- **вероятностный** алгоритм: в качестве номера перезаписываемой строки используется случайное число;
- **FIFO** алгоритм: первая записанная строка будет первой перезаписана;
- **LRU** (Last Recently Used) алгоритм: наименее используемая строка будет заменена новой.

Для повышения производительности процессора вводятся ряд **специальных кэшей.**

- **TLB** (Translation Look-aside Buffers) - это кэш память, используемая MMU для хранения результатов последних трансляций логического адреса в физический. Содержит пары: логический адрес и соответствующий физический адрес.
- **BTC** (Branch Target Cache) - это кэш память, используемая BU для хранения адреса предыдущего перехода и первой инструкции, выполненной после перехода. Имеет целью без задержки заполнить конвейер инструкцией, если переход уже ранее состоялся. BTC может значительно повысить производительность процессора, учитывая время, которое он бы простаивал в ожидании заполнения конвейера после перехода.

Согласование кэшей в мультипроцессорных системах. Если несколько процессоров подсоединены к одной и той же шине адреса и данных и разделяют одну и ту же внешнюю память, то должен быть реализован определенный следящий механизм (snooping) для того, чтобы все внутрипроцессорные кэши всегда содержали **одни и те же** данные.

Рассмотрим, например, систему, содержащую два процессора, каждый из которых может брать управление общей шиной. Если процессор 1, управляющий в данный момент шиной, записывает в ячейку памяти, которая кэширована процессором 2, то данные в кэше последнего становятся устаревшими. Следящий механизм позволяет второму процессору отслеживать состояние шины адреса, даже если он не является в данный момент главным (т.е. управляющим внешней шиной). Если на шине появился адрес кэшированных данных, то эти данные помечаются в кэше как некорректные. Когда второй процессор станет главным, он должен будет выбрать в случае необходимости обновленные данные из разделяемой памяти.

Если процессор 1, управляющий в данный момент шиной, читает из ячейки памяти, которая кэширована процессором 2, то возможно, что реальные данные находятся в кэше процессора 2 (еще не записаны в память, т.е. реализован кэш с обратной записью). Если это так, то следящий механизм инициирует цикл записи строки кэша процессора 2, содержащей затребованные процессором 1 данные, в разделяемую память. После этого эти данные становятся доступными процессору 1 и цикл чтения процессора 1 продолжается.

Поддержка многозадачности и многопроцессорности

В современных процессорах поддержка многозадачности и многопроцессорности не ограничивается аппаратной частью (вроде рассмотренного выше механизма синхронизации кэшей). Для организации доступа к критическим разделяемым ресурсам необходимо в наборе инструкций процессора предусмотреть специальные инструкции, обеспечивающие доступ к объектам синхронизации. Действительно, сами объекты синхронизации являются *разделяемыми*, а для обеспечения правильного доступа к разделяемым объектам необходимо вводить объекты синхронизации, которые в свою очередь тоже являются разделяемыми и т.д. Для выхода из этого замкнутого круга определяют некоторые “примитивные” объекты синхронизации, к которым возможен одновременный доступ нескольких задач или процессоров за счет использования *специальных инструкций доступа.*

Рассмотрим более подробно случай булевского семафора. Задача или процессор, собирающийся взять управление разделяемым ресурсом, начинают с чтения значения семафора. Если он обнулен, то задача или процессор должны ждать, пока ресурс станет доступным. Если семафор установлен в 1, то задача или процессор немедленно его обнуляют, чтобы показать, что контролируют ресурс. В процессе изменения семафора можно выделить три фазы: **чтение, изменение, запись.** Если на стадии чтения возникнет переключение задач или другой процессор станет главным на шине, то может возникнуть ошибка, так как две задачи или два процессора контролируют один и тот же ресурс. Аналогично, если переключение контекста произойдет между циклом чтения и записи, то два процесса могут взять семафор, что тоже приведет к системной ошибке. Для решения этой проблемы большинство процессоров имеют инструкцию, выполняющую неделимый цикл чтение - изменение - запись. Поскольку это одна инструкция, то переключение задач во время операции с семафором невозможно. Так как она производит неделимый цикл, то

процессор, ее выполняющий, остается владельцем шины до окончания операции с семафором.

Влияние требований реального времени на выбор архитектуры процессора

Не все описанные выше приемы повышения производительности процессора ускоряют работу типичной системы реального времени, а иногда и замедляют ее. Этим объясняется то, что на рынке систем для О СРВ значительную долю занимают процессоры с устаревшей CISC архитектурой, имеющие примитивный конвейер и кэш. Рассмотрим основные причины этого положения.

- Доводы (статистический анализ), положенные в основу обоснования RISC архитектуры, оказались не совсем верными для ОСРВ. Часто приложение ОСРВ проводит значительную часть времени в обработке прерываний от внешних устройств. Обработчики прерываний часто пишутся на ассемблере и наличие сложных инструкций в CISC процессоре позволяет уменьшить длину обработчика. Наличие сложных режимов адресации памяти позволяет обойтись меньшим количеством регистров и, соответственно, еще уменьшить обработчик (за счет уменьшения количества сохраняемых/'восстанавливаемых регистров) .
- Увеличение числа регистров в RISC процессорах приводит к увеличению времени на переключение задач и, в частности, на обработку прерываний, поскольку необходимо сохранять/восстанавливать значительное количество регистров.
- Увеличение глубины конвейера приводит к увеличению времени на переключение задач и, в частности, на обработку прерываний, поскольку необходимо дождаться выполнения всех инструкций в конвейере.
- Увеличение глубины конвейера и размеров кэша не всегда приводит к ускорению работы ОСРВ из-за большого количества прерываний (т.е. переключений задач), поскольку конвейеры и кэш не успевают заполниться необходимыми инструкциями и данными.
- Увеличение размеров кэша приводит к увеличению накладных расходов на их синхронизацию в многопроцессорных системах.
- В ряде ситуаций ОСРВ приходится блокировать все конвейеры суперскалярного процессора, кроме одного. Дело в том, что инструкции в таком процессоре могут завершаться не в том порядке, в котором они подавались в процессор. Например, поток инструкций $I_4 \to I_3 \to I_2 \to I_1$ процессор может быть распределен между двумя конвейерами как $I_3 \to I_1 \to$ **конвейер**$_1$, $I_4 \to I_2 \to$ **конвейер**$_2$. Если инструкция I_1 выполняется очень долго (например, в ней делается обращение к памяти), то инструкции I_2, I_4 будут исполнены раньше, чем I_1, I_3. Эта ситуация недопустима при работе с вводом/выводом. Поэтому в суперскалярных процессорах существуют инструкции, переводящие его в *последовательный* режим исполнения инструкций для обеспечения их правильного порядка. Часто эти инструкции фактически блокируют все конвейеры, кроме одного.

Для ускорения обработки прерываний, в некоторых процессорах применяются специальные меры.

- Таблица прерываний может храниться во внутренней памяти процессора, что делает ненужной выборку из внешней памяти (Intel 80960).

- Процессор может включать теневые регистры, что делает ненужным сохранение контекста текущей задачи в простых процедурах обработки прерываний (HP-PA).
- Критические процедуры обработки прерываний могут быть заблокированы в кэше инструкций (Motorola 68060).
- Таблица прерываний может хранить первые инструкции обработчика прерываний, что уменьшает простой конвейера (SPARC).

Лекция 4.2. Организация данных во внешней памяти. Процессоры Motorola 68xxx.

План:

- Организация целочисленных данных
- Организация данных с плавающей точкой
- Пути повышения производительности оперативной памяти
- Процессоры Motorola 68xxx

В системах реального времени достаточно частой является ситуация, когда в одном VME крейте находятся несколько плат, построенных на процессорах разных производителей. Эти платы обычно решают общую задачу по управлению промышленным оборудованием и обмениваются между собой данными через разделяемую память (собственно, это единственный способ обмена, предоставляемый шиной VME, см. раздел 12.1). В этой ситуации необходимо согласование представления данных в памяти каждым из участвующих в обмене процессоров.

Организация целочисленных данных

При доступе к целочисленным данным основным вопросом является способ нумерации байтов в слове.

Если бы в 32-битной архитектуре минимальным адресуемым элементом оперативной памяти являлось бы 32-битное слово, то вопрос о нумерации байтов в слове не вставал бы. В реальной ситуации, когда минимальным адресуемым элементом оперативной памяти является байт (8-ми битное слово), данные большего размера образуются как объединение подряд идущих байт. Выбор нумерации байт в 32 битном (4 байта) слове может быть произвольным, что дает 24 = 4! способа. На практике используются только два: ABCD (называемый big-endian) и DCBA (называемый little-endian).

В **big-endian,** модели байты в слове нумеруются от наиболее значимого к наименее значимому. В **little-endian** модели байты в слове нумеруются от наименее значимого к наиболее значимому. Примеры big-endian процессоров: Motorola бВxxx. PowerPC (по умолчанию), SPARC, пример little-endian процессора: Intel 80x86. Процессоры PowerPC, Intel 80960x, ARM, SPARC (64-битные модели) могут работать как в big-endian режиме, так и в little- endian.

Если процессоры, осуществляющие обмен через разделяемую память, используют разный режим нумерации байтов, то потребуется преобразовывать все полученные или переданные данные. Практически все современные процессоры имеют для этой цели специальную инструкцию.

Организация данных с плавающей точкой

Все современные процессоры поддерживают стандарт ANSI/IEEE 754-1985 в организации данных с плавающей точкой. Необходимо только учитывать, что данные с

плавающей точкой содержат несколько байт, поэтому на них также оказывает влияние способ нумерации байтов в слове.

floating-point single:

- размер - 4 байта,
- знак *s* - бит 31 (1 бит)
- показатель *p* - биты 23... 30 (8 бит)
- мантисса *x* - биты 0... 22 (23 бит)

Нормализованное значение (т.е. при $0 < p < 255$)

$$(-1)^s \bullet 2^{p-127} \bullet 1.x$$

floating-point double:

- размер - 8 байт,
- знак *s* - бит 63 (1 бит)
- показатель *p* - биты 52... 62 (11 бит)
- мантисса *x* - биты 0... 51 (52 бит)

Нормализованное значение (т.е. при $0 < p < 2047$)

$$(-1)^s \bullet 2^{p-1023} \bullet 1.x$$

floating-point quad:

(поддерживается не всеми процессорами)

- размер - 16 байт,
- знак s - бит 127 (1 бит)
- показатель p - биты 112. ..126 (15 бит)
- мантисса x - биты 0... 111 (112 бит)

Нормализованное значение (т.е. при $0 < p < 32767$)

$$(-1)^s .2^{p-16382} -1.x$$

Пути повышения производительности оперативной памяти

Для повышения производительности оперативной памяти памяти применяют несколько приемов.

- **Увеличение ширины шины данных:** переход от SIMM (32 бит шина) к DIMM (64 или 128 бит шина) модулям при построении подсистемы памяти.
- **Введение небольшой статической памяти SRAM для буферизации DRAM модулей:** буферизованные DIMM модули.
- **Введение конвейера в модули DRAM.** Используется та же идея, что и при введении конвейера в процессоры. Внутри модуля памяти находится несколько обрабатываемых обращений к памяти в разной степени готовности (формирование адреса, выбор банка, выборка данных, запись их на внешнюю шину и т.д.). Это

позволяет модулю памяти принимать/'выдавать данные каждый цикл шины (при условии оптимального функционирования конвейера). В силу этого такая память получила название synchronous DRAM (SDRAM). Для такой памяти в качестве времени доступа производители в рекламных целях пишут минимальный цикл шины, что дает фантастические времена доступа менее 10ns (т.е. частота шины более 100MHz). На самом же деле в модулях SDRAM использованы обычные микросхемы памяти, время доступа к которым - около 60ns.

- **“Расслоение” оперативной памяти.** Поскольку процессор обменивается с памятью только блоками размером со строку кэша, то можно разделить этот блок на N частей (N обычно 2, 4, 8) и передать каждую из частей своей подсистеме памяти. В результате получаются N подсистем памяти, работающих параллельно. Если программа требует последовательные адреса памяти (т.е. стратегия предвыборки строки кэша себя оправдывает), то этот подход может в N раз увеличить производительность подсистемы памяти.

Процессоры Motorola 68xxx

Процессоры Motorola 68xxx длительное время доминировали на рынке промышленных компьютерных систем. Сейчас они оттеснены на второе место процессорами PowerPC.

Семейство процессоров Motorola 68xxx (сокращенно M68к) характеризуется простотой реализации как аппаратных, так и программных решений на его базе. Первым процессором семейства являлся хорошо известный M68000. Хотя он появился в 1979 году, он является процессором с полной *32-битной* внутренней архитектурой. Подобно всем последующим процессорам, его линейная организация памяти и единое с памятью пространство ввода/'вывода облегчает аппаратные и программные разработки. Наконец, специальные выводы процессора устанавливаются на каждом цикле шины для того, чтобы различить супервизорское и пользовательское адресные пространства. Если эти выводы декодировать при вычислении адреса, то системные ресурсы будут изолированы от несанкционированного доступа пользовательских программ.

Следующие поколения процессоров M68к, включая M68060, с программной точки зрения вносили только новые режимы адресации памяти и некоторые дополнительные инструкции, и поэтому программируются так же легко, как M68000. Эволюция затронула аппаратную архитектуру процессоров: в их составе появились и развивались конвейеры, кэши, MMU, FPU и т.д.

Основываясь на архитектуре M68k Motorola разработала семейство контроллеров MC683xx, которые разделяют на 3 группы: группу 68000, CPU32 и CPU32+. В этих контроллерах вычислительная мощность M68к сочетается с интегрированными периферийными процессорами, образуя высокопроизводительные контроллеры. Наиболее специализированные, ориентированные на ввод/'вывод контроллеры (68302 и 68360) включают непрограммируемый RISC процессор, управляющий последовательным

коммуникационным каналом. Это дает возможность одному устройству управлять таким сложным последовательным каналом, как Ethernet или ISDN.

Все члены семейства MC683xx имеют межмодульную шину (intermodule bus, IMB). IMB обеспечивает общий интерфейс для всех модулей семейства MC683xx, что позволяет фирме Motorola быстро разрабатывать новые устройства, используя библиотеки существующих модулей.

Основные члены семейства

Каждый процессор в приведенных ниже технических данных обладает всеми возможностями предыдущих процессоров для обеспечения совместимости.

M68000 (1979, 68000 транзисторов, макс, частота: 16.67MHz, 1 MIPS):

- асинхронные передачи данных (кроме области интерфейса 6800)
- *16-битная* шина данных, отсутствует динамическое изменение ширины шины
- *24-битная* шина адреса
- 14 режимов адресации памяти (включая косвенные регистровые)
- 16 32-битных регистров общего назначения
- два уровня привилегий: пользовательский и супервизорский
- два указателя стека для разделения пользовательского и супервизорского стеков
- фиксированная в памяти таблица прерываний (начиная с адреса 0)
- одна неделимая инструкция TAS (Test And Set) для установки семафоров

M68010 (1983, 68000 транзисторов, макс, частота: 16.67MHz, 1 MIPS):

- перемещаемая таблица прерываний
- поддержка механизма виртуальной памяти

M68020 (1984, 195000 транзисторов, макс, частота: 25MHz, 5 MIPS):

- асинхронные передачи данных
- *32-битная* шина данных, динамическое изменение ширины шины (адаптируется к 8-ми, 16-ти и 32-х битным обменам с внешними устройствами)
- *32-битная* шина адреса
- 18 режимов адресации памяти (включая косвенные через память)
- *256-байт* кэш инструкций
- *3-х стадийный* конвейер, что позволяет одновременно обрабатывать до трех слов одной операции или три последовательные инструкции
- интерфейс сопроцессора, что позволяет подключить внешнее FPU (MC68881 или MC68882) и MMU (MC68851)
- дополнительный указатель стека для разделения аппаратных и программных прерываний

- две дополнительных неделимых инструкции: CAS и CAS2 (Compare And Swap 32 или 64 бита) для установки семафоров
- новые инструкции для работы с битовыми полями

M68030 (1986, 300000 транзисторов, макс, частота: 50MHz, 8 MIPS):

- Гарвардская архитектура
- два различных кэша: *256-байт* кэш инструкций + *256-байт* кэш данных
- асинхронные передачи данных
- синхронный интерфейс, что позволяет осуществлять блочные (burst) передачи в/'из кэша
- MMU, позволяющее работать со страницами размером от 256 байт до 256 килобайт

M68040 (1989, 1200000 транзисторов, макс, частота шины: 40MHz, 8 MIPS, 3.5Mflops):

- синхронные передачи данных
- отсутствует динамическое изменение ширины шины
- частота процессора равна удвоенной частоте шины (максимально 80MHz)
- *6-ти стадийный* конвейер
- FPU
- *4-килобайт* кэш инструкций + *4-килобайт* кэш данных
- механизм синхронизации шины (bus snooping) (арбитраж шины должен быть внешним, процессор имеет вывод запроса шины), это обеспечивает согласование кэшей в многопроцессорных системах

M68060 (1994, 2500000 транзисторов, макс, частота шины: 66MHz, 100 MIPS):

- суперконвейерный, суперскалярный 32 битный гибридный CISC-RISC процессор
- набор инструкций M68040 реализован аппаратной логикой, а не микрокодом
- основные устройства: буфер инструкций, 4-х стадийное конвейерное устройство предвыборки, два 4-х стадийных конвейерных целочисленных устройства, устройство переходов, FPU повышенной точности
- кэш переходов (BTC)
- поток инструкций разделяется на два конвейера на FIFO стадии
- устройство предвыборки преобразует входной поток M680x0 инструкций, имеющих переменную длину, в поток RISC инструкций фиксированной длины
- M68060 может может исполнять за один цикл 4 инструкции M680x0: две целочисленных инструкции, одну инструкцию перехода и одну инструкцию с плавающей точкой; этот параллелизм обеспечивает высокую скорость исполнения даже для кода, не перекомпилированного специально для M68060
- *4-килобайт* кэш инструкций + *4-килобайт* кэш данных
- автоматическое уменьшение потребляемой мощности: внутренние функциональные блоки автоматически выключаются, если они не используются в течении ряда циклов

M68302 (микроконтроллер группы 68000):

- IMP (Integrated Multiprotocol Processor)
- M68302 состоит из процессора M68000, System Integration Block (SIB) и Communication Processor (CP)
- SIB содержит контроллер DMA, два 16-битных таймера общего назначения, контроллер памяти и контроллер прерываний
- CP является выделенным RISC процессором, обслуживающим 6 последовательных портов, и отвечает за работу с последовательными каналами по выбранному пользователем протоколу (ISDN, UART, HDLC, BSC и другие)

M68360 (или QUICC) (микроконтроллер группы CPU32+):

- M68360 состоит из процессора CPU32+, SIM60 (System Integration Module) и CPM (Communication Processor Module)
- процессор CPU32+ является процессором M68020 без кэша и сопроцессорного интерфейса; система команд процессора M68020 расширена CPU32+ специфическими инструкциями табличной интерполяции
- SIM60 интегрирует основные устройства общего назначения, которые могут быть полезны в любых 32-битных процессорных системах: синтезатор частоты, таймеры-будильники, таймер периодического прерывания, контроллер памяти, способный без дополнительной логики управлять 32-битными DRAM
- CPM содержит 2 DMA контроллера, 4 таймера общего назначения, контроллер прерываний, 7 коммуникационных контроллеров, управляющих 7-ю последовательными физическими каналами
- поддерживаются протоколы UART, ISDN, HDLC, BSC, AppleTalk
- протокол Ethernet поддерживается в версии MC68EN360

Программная модель

Прикладной программе доступны 16 регистров общего назначения и несколько служебных регистров.

Регистры d0 - d7 (регистры данных):

Могут содержать целочисленные данные следующих типов

1. Бит (M68020 и выше, только инструкции, работающие с битовыми полями)
2. Двоично-закодированные десятичные числа (BCD); байт содержит одну цифру, существуют инструкции, работающие с двумя цифрами в одном байте
3. Байт (8 бит); при записи в регистр старшая часть не используется и не изменяется
4. Слово (16 бит); при записи в регистр старшая часть не используется и не изменяется
5. Длинное слово (32 бит)
6. Счетверенное слово (64 бит); используются любые два регистра, над такими операндами определена только инструкция пересылки (MOVEM).

Регистры aO - a7 (регистры данных):

Могут содержать целочисленные данные следующих типов

1. Слово (16 бит); при записи в регистр старшая часть заполняется знаковым битом источника
2. Длинное слово (32 бит)

Регистр pc (program counter):

содержит адрес следующей инструкции. При записи в этот регистр происходит переход по адресу, который был записан.

Регистр кода условия ccr (condition code register):

является частью регистра статуса (status register, SR), который не доступен пользовательской программе как регистр. Коды условия устанавливаются по результату арифметических операций или специальными инструкциями, и используются в командах условного перехода.

Поддерживаются следующие режимы адресации памяти (в терминах языка C).

- $*(A_n + +)$
- $*(- - A_n)$
- $*(B_n + d + X_n)$
- $*(*(B_n + d) + X_n + o)$
- $*(*(B_n + d + X_n) + o)$

где

- $\mathbf{A_n}$ - адресный регистр **a0 - a7**
- $\mathbf{B_n}$ - адресный регистр **a0** - **a7** или **pc**
- $\mathbf{X_n}$ - отсутствует или регистр **a0** - **a7** или **d0 - d7;** у M68020 и выше может быть $\mathbf{X_n}$ * **s, s= 1, 2, 4, 8**
- **d** - константа со знаком (8, 16, 32 бит), включая 0
- **o** - константа со знаком (16, 32 бит), включая 0

Инструкции процессоров Motorola 68xxx имеют от нуля до трех операндов. Большинство инструкций (пересылки, арифметические, логические) имеют два операнда, один из которых не изменяется в операции (источник), а другой является результатом операции (приемник). Обычно используется синтаксис, в котором источник является левым операндом, а приемник - правым.

Существует несколько инструкций, неявно использующих указатель стека a7. Это стековые операции и вызовы функций.

План:

- Основные члены семейства процессора Intel 80x86
- Программная модель процессора Intel 80x86
- Основные члены семейства процессора PowerPC
- Программная модель процессора PowerPC

Процессоры Intel 80x86

Процессоры Intel 80x86 доминируют на рынке персональных компьютеров. На рынке промышленных систем их роль незначительна.

Первый представитель семейства - 18086, появился в 1979г. Это был 16-ти битный процессор, обеспечивающий совместимость с предыдущими 8-ми битными процессорами 18080 и 18085. Это наложило определенный отпечаток на программную архитектуру 18086:

- устаревший набор инструкций,
- множество нелогичных ограничений на операнды.

Основные общие черты семейства i80x86

- пространство ввода/'вывода отделено от пространства памяти,
- сегментная организация памяти,
- малое число регистров,
- невзаимозаменяемость регистров (много инструкций, неявно использующих жестко закрепленные регистры).

Процессор i80386 был первым 32-х битным процессором в семействе и уже мог работать с плоской моделью памяти. Однако, этот процессор сохранил программную совместимость с предыдущими членами семейства. Это еще больше увеличило его сложность и законсервировало устаревшую программную архитектуру i8086.

Основные члены семейства

Каждый процессор в приведенных ниже технических данных обладает всеми возможностями предыдущих процессоров для обеспечения совместимости.

i80386 (1987, 275000 транзисторов, 33MHz, 8 MIPS):

- *32-бит* шина данных и *32-бит* шина адреса
- 8 32-битных регистров общего назначения
- динамическое изменение ширины шины (16 или 32 бит)
- пространство памяти отделено от 64Кб пространства ввода/'вывода
- буфер предвыборки 16 байт
- MMU (страничное или (и) сегментное) с кэшем результатов трансляции; может работать в 3-х режимах:

- *real mode :* 24 бит сегментный адрес, нет трансляции
- *protected mode :* 32 бит адрес, страничное или (и) сегментное преобразование
- *virtual 8086 mode :* 24 бит сегментный адрес, страничная трансляция

- внешнее FPU i80387
- внешний контроллер (внешней) кэш памяти i82385
- поддержка многозадачности посредством регистра задачи и дескрипторов задач

180486 (1989, 1200000 транзисторов, частота шины 33MHz, 20 MIPS):

- внутренняя частота процессора равна либо частоте шины (DX 33MH/.). либо удвоенной частоте шины (DX2 66MII/). либо утроенной частоте шины (DX4 100MHz)
- единый 8Кб (16Кб) кэш инструкций и данных (с поддержкой блоковых передач (burst access))
- кэш сквозной записи (write through) / обратной записи (write back)
- механизм синхронизации кэшей в многопроцессорных системах (bus snooping)
- 32 байт буфер предвыборки
- встроенное FPU (без конвейера)
- 5-ти стадийный конвейер, использующий таблицу регистров (register scoreboard)

Pentium (1993, 3100000 транзисторов, частота шины 50/60/66MHz, 100 specint92, 80 specfp92):

- внутренняя частота процессора равна либо частоте шины (60. 66MII/). либо 1.5 частоты шины (75.90. 100MH/.). удвоенной частоте шины (120.133MII/.). 2.5 частоты шины (150.166MII/.). либо утроенной частоте шины (180. 200MII/.)
- 64-бит внешняя шина данных, но 32-битная внутренняя архитектура
- суперскалярная архитектура: 2 IU
- аппаратное декодирование
- устройство предсказания переходов (BU)
- 5-ти стадийный конвейер в IU, 8-ми стадийный конвейер в FPU
- Гарвардская архитектура: 8Кб кэш инструкций и 8Кб кэш данных
- механизм синхронизации кэшей MESI (Modified, Exclusive, Shared, Invalid), используемый в PowerPC
- гибридная CISC/RISC архитектура

Pentium Pro (1996, 5500000 транзисторов, 150+MHz, 366 specint92, 283 specfp92):

- суперскалярный процессор: до 5 инструкций за цикл
- 14-ти стадийные конвейеры (2 для IU и 1 для FPU)
- Гарвардская архитектура: 16Кб кэш инструкций и 8Кб кэш данных
- архитектура DIB (Dual Independent Bus): одна шина связывает процессор с кэш памятью второго уровня, а вторая - с **ОЗУ**

- 256(512)Кб встроенный кэш второго уровня; расположен на отдельном кристалле, но в том же корпусе, связан с процессором 64-битной шиной, работающей на частоте процессора
- исполнение инструкций не по порядку
- спекулятивное исполнение: выполнение инструкций за точкой ветвления
- переименование регистров (register renaming)

Pentium-ll (1997, 7500000 транзисторов, 233+MHz):

- Pentium Pro ядро с архитектурой DIB
- суперскалярный процессор: 2IU и 2FPU
- Гарвардская архитектура: 16Кб кэш инструкций и 16Кб кэш данных
- 512Кб встроенный кэш второго уровня; расположен на отдельном кристалле, связан с процессором 64-битной шиной, работающей на половине частоты процессора (для модификации Pentium-II Xeon - на частоте процессора)

Pentium-lll (1999, 450+MHz):

- Pentium ll с дополнительными мультимедийными инструкциями
- 512Кб встроенный кэш второго уровня; расположен на отдельном кристалле, связан с процессором 64-битной шиной, работающей на половине частоты процессора (для модификации Pentium-lll Xeon - на частоте процессора)

P7 (Intel и HP, 2000):

- 64-бит архитектура
- RISC процессор с набором команд PA-RISC, способен выполнять инструкции i8086

Pentium-4 (2000, 1400+MHz):

- Pentium III с увеличенной частотой шины между L2 кэшем и внешней памятью.

Программная модель

Прикладной программе доступны 8 регистров общего назначения и несколько служебных регистров.

Регистры eax, ebx, ecx, edx:

Могут содержать целочисленные данные следующих типов

1. Бит (только инструкции, работающие с битовыми полями)
2. Двоично-закодированные десятичные числа (BCD); байт содержит одну цифру, существуют инструкции, работающие с двумя цифрами в одном байте
3. Байт (8 бит); при записи в регистр старшая часть не используется и не изменяется; возможен доступ к битам 0... 7 и битам 8... 15 соответственно по именам **al, Ы, cl, dl** и **ah, bh, ch, dh**
4. Слово (16 бит); при записи в регистр старшая часть не используется и не изменяется; обращение осуществляется по имени **ax, bx, cx, dx** соответственно
5. Длинное слово (32 бит)

Регистры esi, edi, ebp, esp:

Могут содержать целочисленные данные следующих типов

1. Бит (только инструкции, работающие с битовыми полями)
2. Двоично-закодированные десятичные числа (BCD); байт содержит одну цифру, существуют инструкции, работающие с двумя цифрами в одном байте
3. Слово (16 бит); при записи в регистр старшая часть не используется и не изменяется; обращение осуществляется по имени **si, di, bp, sp** соответственно
4. Длинное слово (32 бит)

Регистр кода условия ccr (condition code register):

является частью регистра флагов (EFLAGS), который не доступен пользовательской программе как регистр. Коды условия устанавливаются по результату арифметических операций или специальными инструкциями и используются в командах условного перехода.

Следующие регистры доступны пользовательской программе, но не используются в прикладных программах для UNIX систем (поскольку они работают в плоской модели памяти):

Сегментные регистры gs, fs, es, ds, cs, ss:

содержат 16-бит селекторы сегментов, неявно участвуют при формировании адреса

Все режимы адресации памяти процессоров Intel 80x86 можно записать одной формулой: адрес ячейки памяти есть

base + index * scale + displacement

где

- **base** - базовый регистр: **eax, ebx, ecx, edx, esi, edi, ebp, esp**
- **index** - индексный регистр: **eax, ebx, ecx, edx, esi, edi, ebp**
- **scale** - целая константа 1, 2, 4, 8
- **displacement** - смещение 8 или 32 бит

Любой из элементов адреса может отсутствовать (с одним исключением: если отсутствует **index,** то должен отсутствовать и **scale).**

Инструкции процессоров Intel 80x86 имеют от нуля до трех операндов (явных или неявных). Большинство инструкций (пересылки, арифметические, логические) имеют два операнда, один из которых не изменяется в операции (источник), а другой является результатом операции (приемник). Фирма Intel использует синтаксис, в котором приемник является левым операндом, а источник - правым. Но большинство ассемблеров в UNIX системах использует синтаксис a la Motorola 68xxx: левый операнд - источник, правый - приемник.

У большинства двухоперандных инструкций один из операндов (источник или приемник) может быть регистром или ячейкой памяти, другой тогда может быть регистром или непосредственным значением. Это позволяет разделить двухоперандные инструкции на следующие группы:

- регистр - регистр
- регистр - память
- память - регистр
- непосредственное значение - регистр
- непосредственное значение - память

Существуют инструкции, (неявно) осуществляющие операции типа "память - память". Это строковые инструкции и операции со стеком.

Инструкции, неявно использующие жестко закрепленные регистры:

- умножение и деление с двойной точностью
- ввод/вывод
- работа со строками
- циклы
- сдвиги
- операции со стеком (включая вызовы функций)
- инструкция трансляции

В 32-битном режиме работы процессора большинство инструкций имеют операнды размером 8 или 32 бит; использовать операнды размером 16 бит можно при использовании специального префикса инструкции. Однако, изменить размер смещения в адресе (8 или 32 бит) таким способом невозможно.

Процессоры PowerPC

Архитектура PowerPC (Performance Optimized With Enhanced Rise Personal Computer) была разработана совместно IBM, Motorola и Apple. В настоящее время она доминирует на рынке промышленных систем.

Архитектура PowerPC определяет три архитектурных уровня:

- ***UISA*** **(User Instruction Set Architecture)** определяет уровень архитектуры, которому должно удовлетворять пользовательское программное обеспечение. UISA задает программную модель и модель памяти для пользовательских программ. Именно, UISA определяет доступный прикладной программе набор инструкций, набор регистров, типы данных, соглашения о хранении чисел с плавающей точкой в памяти, модель исключений, видимую прикладной программой.
- ***VEA*** **(Virtual Environment Architecture)** определяет дополнительный уровень архитектуры, которому должно удовлетворять пользовательское программное обеспечение, выходящее за рамки обычных требований прикладных программ. VEA задает модель памяти для окружений, в которых множество устройств могут получать доступ к памяти, определяет модель кэша и инструкции управления кэшем.
  •
- ***OEA*** **(Operating Environment Architecture)** определяет уровень архитектуры, которому должно удовлетворять супервизорское программное обеспечение (операционные системы). OEA определяет модель управления памятью, регистры

уровня суперпользователя, требования к синхронизации процессов, модель исключений.

Эти спецификации позволяют разрабатывать новые процессоры семейства, сохраняя программную совместимость с существующими и будущими процессорами PowerPC. Впервые в истории разработки процессоров важнейшие усилия по спецификации были предприняты до появления первого процессора.

В дополнение к этим архитектурным определениям три производителя (IBM, Motorola, Apple) разработали эталонную *платформу* для разработки плат на основе PowerPC. *CHRP* (Common Hardware Reference Platform) является открытой спецификацией для разработки компьютерных систем на основе PowerPC. Отметим два важнейших аспекта этой спецификации. Во-первых, спецификация описывает устройства, интерфейсы и форматы данных, требуемые для разработки и построения законченной компьютерной системы. Она описывает методы абстрагирования аппаратных деталей от операционной системы. Более того, CHRP предписывает использовать вторую PCI шину, так, что шина PowerPC разделяется только L2 кэшем, контроллером памяти и мостом PCI. Это позволяет разработчикам использовать новые процессоры PowerPC без изменений в остальной части платы. Во-вторых, спецификация описывает эталонную реализацию операционной системы, согласованную с известными операционными системами для PowerPC: AIX (IBM), Solaris (Sun), Windows NT (Microsoft).

Основные члены семейства

Все перечисленные ниже процессоры удовлетворяют стандарту на PowerPC и отличаются набором аппаратных устройств.

PowerPC 603 (1993, 1600000 транзисторов):

- 8Кб кэш инструкций + 8Кб кэш данных (32-бит Гарвардская архитектура)
- 5 независимых исполняющих устройств: 1 IU + 1 FPU + 1 BU + 1 load /store unit + 1 system register unit
- FPU конвейеризировано, так, что инструкции сложения и умножения одинарной точности могут обрабатываться каждый цикл
- 4-х стадийный конвейер
- MMU, выполняющее сегментную и страничную трансляцию адреса из 52битного логического адреса в 32-битный физический адрес
- два 64-входовых TLB
- потребление 3Вт на частоте 80MHz, четыре программно контролируемых режима экономии энергии

PowerPC бО3е (1993, 1600000 транзисторов):

Вариант PowerPC 603 для настольных систем, частота до 240MHz

PowerPC 604 (1994, 3600000 транзисторов):

- суперскалярный процессор (4 инструкции за цикл)
- 16Кб кэш инструкций + 16Кб кэш данных (32-бит Гарвардская архитектура)
- 6 независимых исполняющих устройств: 3 IU + 1 FPU + 1 BU + 1 load /'store unit

- 6-ти стадийный конвейер
- два 128-входовых TLB
- возможность последовательного выполнения инструкций
- переименование регистров (register renaming)
- *динамическое* предсказание переходов
- *поддержка многопроцессорности* посредством специального протокола синхронизации кэшей
- потребление 10Вт на частоте 100MHz, один программно контролируемый режим экономии энергии

PowerPC 604e (1994, 3600000 транзисторов):

Вариант PowerPC 604 для настольных систем, частота до 300MHz

PowerPC 620 (1995, 7000000 транзисторов, на частоте 133MHz 225 specint 92, 300 specfp 92):

- 128-бит шина данных, 40-бит шина адреса, 64-бит регистры
- суперскалярный процессор (6 инструкций за цикл)
- *32Кб* кэш инструкций + *32Кб* кэш данных (64-бит Гарвардская архитектура)
- 6 независимых исполняющих устройств: 2 IU + 1 FPU + 1 BU + 1 load /'store unit + 1 complex unit
- 5-ти стадийный конвейер
- 2-х уровневое MMU, первичное MMU имеет 64-входовый TLB, вторичное MMU имеет 128-входовый TLB
- для ускорения переходов count register (ctr) имеет теневой регистр, в который он переименовывается во время перехода
- интегрированный контроллер L2 кэша
- потребление 30Вт на частоте 100MHz, один программно контролируемый режим экономии энергии

PowerPC G3 (1997, не более 30000000 транзисторов, частота 300MHz):

развитие PowerPC 620, интегрированный L2 кэш

PowerPC G4 (1998, не более 50000000 транзисторов, частота 400MHz):

развитие PowerPC G3

IBM PowerPC 403 (1994):

- PowerPC микроконтроллер
- 2Кб кэш инструкций + 1Кб кэш данных (32-бит Гарвардская архитектура)
- периферийные интерфейсные устройства: интерфейс шины, DMA контроллер, контроллер прерываний (на 6 запросов), последовательный порт
- до восьми интерфейсов банков памяти и устройств ввода/'вывода
- 4 таймера
- низкое потребление энергии

Motorola MPC500 (1994, 40 MIPS, 25MHz):

- PowerPC микроконтроллер
- 4Кб кэш инструкций
- 4 Кб SRAM
- 4 независимых исполняющих устройств: 1 IU + 1 FPU + 1 BU + 1 load /'store unit
- контроллер прерываний на 32 запроса (время задержки links на частоте 25MHz)
- контроллер памяти, способный управлять 12 микросхемами памяти
- встроенный автотест
- очень низкое потребление энергии (530mW)
- Motorola PowerQUICC (1996, 52 MIPS, 40MHz):
- улучшенная версия QUICC (M68360)
- ядро CPU32+ заменено на ядро MPC500
- 4 высокоскоростных последовательных коммуникационных канала контролируются выделенным RISC коммуникационным процессором, работающим независимо от основного процессора
- PowerQUICC по сравнению с QUICC имеет контроллер PCMCIA 2.01 и аналог процессора DSP

Программная модель

Прикладной программе доступны следующие регистры.

32 целочисленных регистра общего назначения (РОН) г0 - г31:

содержат слово (32/64 бит)

32 регистра с плавающей точкой ГО - f31:

содержат значение с плавающей точкой (64 бит)

cr (condition register):

это 32-битный регистр, разделенный на восемь 4-х битных полей **cr0-cr7.** Поля регистра **cr** могут быть установлены одним из следующих способов.

- Указанное поле регистра **cr** может быть установлено с помощью инструкции пересылки **mtcrf** в **cr** из РОН.
- Указанное поле регистра **cr** может быть установлено с помощью инструкции пересылки **mcrf** в **с г** из другого поля **cr.**
- Указанное поле регистра **хег** может быть скопировано в регистр **cr** с помощью инструкции **mcrxr.**
- Указанное поле регистра **fpscr** может быть скопировано в регистр **cr** с помощью инструкции **mcrfs.**
- Поля регистра **cr** могут быть изменены с помощью логических операций, определенных над полями **cr.**

- **crO** может быть неявным результатом целочисленной инструкции. Все целочисленные инструкции имеют бит **Rc;** если его установить, то биты в **crO** будут установлены сравнением результата инструкции с нулем.
- **crl** может быть неявным результатом инструкции с плавающей точкой и указывать на статус исключения с плавающей точкой. Все инструкции с плавающей точкой имеют бит **Rc;** если его установить, то биты в **crl** будут установлены копированием соответствующих битов из **fpscr.**
- Указанное поле регистра **cr** может быть результатом целочисленной или вещественной инструкции сравнения.

fpscr (Floating Point Status and Control Register):

это 32-битный регистр, содержащий все биты сигналов исключений для операций с плавающей точкой, биты суммарных исключений, биты разрешения исключений, биты управления округлением, необходимые для удовлетворения стандарту IEEE 754.

xer register:

это 32-битный регистр, содержащий флаги переполнения и переносов для целочисленных операций. Также содержит число байтов, которые нужно передать в инструкциях Load String Word Indexed **(lswx)** или Store String Word Indexed **(stswx)**

lr (link register):

это 32/64-битный регистр, содержащий адрес перехода для инструкций Branch Conditional to Link Register **(bclr)** и Branch and Link **(Ы).** Содержит адрес вызвавшей функции сразу после вызова.

ctr (count register):

это 32/64-битный регистр, содержащий счетчик цикла, который может быть декрементирован в течение выполнения надлежащим образом закодированных инструкций перехода. Регистр ctr может также содержать адрес перехода для инструкции Branch Conditional to Count Register (bcctr).

Поддерживаются следующие два режима адресации памяти.

- адрес = (rA|10) + offset (включая offset=0)
- адрес = (rA|0)+гВ

где обозначено

- (rA|0) - РОН г1...г31 , если rA не равно r0, иначе 0
- offset - 16-бит смещение (знаково расширяемое)
- rB - РОН r0... r31

Все инструкции (за исключением load/store) имеют операнды в регистрах и потому размер всех операндов равен размеру слова (32/64 бит). Подавляющее большинство инструкций - трехоперандные, хотя существует инструкция с 6-ю операндами.

В большинстве арифметических инструкций можно установить код условия сг0 по результату.

Отметим, что несмотря на принадлежность к классу RISC процессоров, PowerPC имеет даже больше инструкций, чем классический CISC процессор Motorola 68xxx.

План:

- Процессоры SPARC
- Процессоры Intel 80960x
- Процессоры ARM
- Архитектура системной шины

Процессоры SPARC

В 1987 году Sun Microsystems аннонсировала SUN-4 - первую компьютерную систему, основанную на новой RISC CPU архитектуре *SPARC* (Scalable Processor ARChitecture). В отличие от многих других процессоров, SPARC позиционировался как *открытая архитектура,* которую могут использовать все.

SPARC процессоры используют Берклевскую архитектуру, основанную на регистровых окнах. Внутренние регистры образуют блоки с частичным перекрытием. Исследователи Беркли предложили использовать внутренний стек для того, чтобы избежать сохранения и восстановления регистров во внешней памяти. Основной целью было ускорение вызовов процедур за счет минимизации числа обращений к памяти, требуемых для передачи параметров и получения результата.

В каждый момент времени функция может иметь доступ к 32 регистрам: 8 *global registers* (r0 - r7, общие для всех функций) и окну из 24 регистров (r8 - r31). Регистровые окна перекрываются по 8-ми регистрам. В каждой функции выделяют 8 in *registers* (r24 - r31 или i0 - i7) (совпадают с *out registers* в процедуре, вызвавшей данную), 8 *local registers* (r16 - r23 или l0 - l7) (доступны только данной процедуре) и 8 *out registers* (r8 - r15 или o0 - o7) (совпадают с *in registers* в любой процедуре, вызванной данной). При вызове функции происходит переключение окон, так, что вызванная функция получает новый набор регистров l0 - l7, o0 – o7 и разделяет регистры i0 - i7 с вызвавшей функцией (где они адресовались как o0 - o7). Поэтому в регистрах o0 - o7 удобно размещать параметры для вызванной процедуры, в регистрах l0 - l7 - локальные переменные.

Размер каждого окна - 16 регистров (8 *out* + 8 *local* регистров). Фирма SUN специфицировала, что число окон может быть от 2 до 32 (в зависимости от реализации). Тем самым общее число регистров в окнах - от 40 (2x16+8) до 520 (32x16+8).

Если глубина вложенности функций превышает число окон, то процессор генерирует прерывание и операционная система должна сохранить часть окон в памяти.

Существует возможность вызвать функцию без переключения окон.

Другой важной особенностью SPARC архитектуры являются delay slots.

Основные члены семейства

Реализация всех SPARC процессоров удовлетворяет версии архитектуры SPARC с тем или иным номером. Важнейшей функцией SPARC International Compatibility and Compliance

Committee является выработка и публикация SPARC Compliance Definitions, а также инструкций по переходу от одного определения к другому.

Каждый процессор в приведенных ниже технических данных обладает всеми возможностями предыдущих процессоров для обеспечения совместимости.

SPARC (1987, 55000 транзисторов, 33 MHz, 20 MIPS):

- *136* 32-бит регистров
- 32-бит шина адреса и данных
- 4-х стадийный конвейер
- 2 исполняющих устройства: 1 IU (data + address) + 1 shift unit
- внешний CMU (Cache controller and MMU) и FPC (Floating Point Controller)
- всего 50 инструкций, каждая исполняется за 1 цикл

MICROSPARC-II (1994, 85 MHz, 85 MIPS, 64 Specint 92, 55 Specfp 92):

- 32-бит SPARC V8 архитектура
- 16Кб кэш инструкций + 8Кб кэш данных (32-бит Гарвардская архитектура)
- 3 исполняющих устройства: 1 IU + 1 FPU + 1 MMU
- интерфейс сопроцессора
- встроенная логика для поддержки DRAM и ввода/вывода

SuperSPARC-ll (1995, 3100000 транзисторов, 90 MHz, 148 Specint 92, 143 Specfp 92):

- 32-бит SPARC V8 архитектура
- 3 инструкции за цикл
- 8-ми портовый 32x32 файл регистров
- 20Кб кэш инструкций + 16Кб кэш данных (32-бит Гарвардская архитектура)
- 64-входовый TLB (для обоих кэшей)
- 64-бит шина данных
- 4-х стадийный конвейер
- *7* исполняющих устройств: 3 IU + 1 FPU multiply + 1 FPU divide + 1 BU + 1 load/'store unit
- биты предсказания переходов

UltraSPARC-II (1996, 250 MHz, 400 Specint 92, 450 Specfp 92):

- *64-бит* SPARC V9 архитектура
- *4* реально исполняемых инструкции за цикл
- суперскалярный процессор: 6 исполняющих устройств: 2 IU + 2 FPU + 1 BU + 1 load/'store unit
- 9-ти стадийный конвейер
- предсказание переходов
- register scoreboarding
- bi-endian (big- и little-endian порядок байтов)

- 64-бит виртуальный адрес и целочисленные данные
- спекулятивное исполнение
- многоуровневая обработка прерываний, организованных в стек
- встроенная мультимедийная поддержка для 2-х и 3-х мерной графики и новый оптимизированный набор мультимедиа инструкций
- поддержка сильно связанных процессоров (до 4-х процессоров могут разделять одну шину адреса)

Программная модель

Прикладной программе доступны следующие регистры.

32 целочисленных регистра общего назначения (РОН) r0 - r31:

содержат слово (32 бит). Любая пара rN, r(N + 1) при четном N может содержать данные длиной 64 бит. Регистры организованы в виде окон по 24 регистра с перекрытием по 8-ми регистрам.

Выделяют

g0 - g7: это регистры **r0 - r7,** являются общими для всех функций (т.е. не участвуют в переключении регистровых окон). Регистр g0 - специальный: при чтении из него всегда читается 0, при записи в него ничего не происходит (запись не производится, из регистра всегда читается 0)

o0 - o7: это регистры **r8 – r15** *(out registers);* являются *in registers* для любой функции, вызванной данной (т.е. доступны вызванной функции); регистр o7 - специальный, инструкция CALL записывает в него свой собственный адрес (т.е. адрес возврата - 8)

i0 - i7: это регистры **r1б - r23** *(local* registers); не доступны ни функции, вызвавшей данную, ни функциям, вызванным данной

i0 – i7*:* это регистры **r24 - r31** *(in registers*); являются *out registers* для функции,вызвавшей данную (т.е. доступны вызвавшей функции и не доступны любой функции, вызванной данной)

32 регистра с плавающей точкой f0 - f31:

содержат значение с плавающей точкой (32 бит). Любая пара fN, f(N+l) при четном N может содержать данные длиной 64 бит (floating point double), любая четверка fN, f(N+1), f(N+2), f(N+3) при кратном 4-м N может содержать данные длиной 128 бит (floating point quadre).

y (multiply/divide register):

содержит старшую часть произведения (в инструкциях умножения) или старшую часть делимого (в инструкциях деления); читается и записывается инструкциями RDY и WRY

Целочисленные коды условия icc (integer condition code):

являются частью PSR (Processor State Register), который не доступен пользовательской программе как регистр. Коды условия устанавливаются по

результату арифметических операций (если это указано в инструкции) или специальными инструкциями и используются в командах условного перехода.

Поддерживаются следующие два режима адресации памяти.

- адрес = **rA + offset** (включая **offset=0)**
- адрес = **rA + rB**

где обозначено

- rA, rA - РОН r0... г31
- offset - 13-бит смещение (знаково расширяемое)

Все инструкции (за исключением load/store) имеют операнды в регистрах и потому размер всех операндов равен размеру слова (32 бит). Подавляющее большинство инструкций - трехоперандные.

В большинстве арифметических инструкций можно установить коды условия icc по результату.

Процессоры Intel 80960x

Процессоры Intel 80960x занимают значительное место на рынке встраиваемых компьютеров. На их базе построено значительное количество управляющих систем для принтеров, SCSI контроллеров, сетевых коммутаторов.

Процессоры Intel 80960x были специально разработаны для встраиваемых систем. Являются 32-битными RISC процессорами с чертами, присущими CISC процессорам. Как RISC процессоры имеют:

- значительное число регистров (32)
- архитектуру load /'store
- фиксированный формат инструкций (все инструкции, кроме load/store, имеют размер 4 байта)

Как CISC процессоры имеют:

- переменную длину инструкций (инструкции load/store могут иметь длину 8 байт, все остальные имеют длину 4 байта)
- значительное число режимов адресации памяти (11)

Специально для систем реального времени имеют:

- встроенный программируемый контроллер прерываний, подключенный к 8-ми внешним линиям прерывания •
- встроенные подпрограммы (микрокод) для автоматического переключения контекста при прерывании
- встроенное ОЗУ (обычно 1Кб) для хранения таблицы прерываний
- встроенный кэш регистров, ускоряющий переключение контекста
- встроенные программируемые таймеры (ряд моделей)

Процессоры Intel 80960x особенно эффективны в приложениях, требующих обслуживания большого количества прерываний и пересылки больших объемов данных, таких, как обработка графической информации и коммуникационные задачи.

Основной особенностью процессоров Intel 80960x является использование автоматически сохраняемых при вызове процедур и переключениях контекста наборов регистров, что приводит к минимизации числа обращений к памяти. В каждый момент времени функция может иметь доступ к 32 регистрам: 16-ти *global registers* (gO - gl5), общих для всех функций, и набору из 16-ти *local registers* (l0 - l15), доступных только этой функции. При вызове процедуры происходит переключение наборов локальных регистров, так, что вызванная функция получает новый набор регистров **l0 - l15** и разделяет регистры **g0** – **g15** с вызвавшей функцией. Поэтому в регистрах **g0** - **gl5** удобно размещать параметры для вызванной процедуры, в регистрах **l0 - l15** - локальные переменные. Внутри процессора есть кэш для наборов локальных регистров (обычно на 4 набора). Если глубина вложенности функций превышает размер кэша, то процессор генерирует прерывание и операционная система должна сохранить часть наборов регистров в памяти. Отметим, что существует возможность вызвать функцию без переключения наборов локальных регистров.

Основные члены семейства

Каждый процессор в приведенных ниже технических данных обладает всеми возможностями предыдущих процессоров для обеспечения совместимости.

1809б0Кх (1988, 20 MHz, 7.5 MIPS):

- 32-бит архитектура с 4Gb адресным пространством
- 32-бит *мультиплексированная* шина адреса и данных, динамическое изменение ширины шины (адаптируется к 8-ми, 16-ти и 32-х битным обменам с внешними устройствами)
- 512 байт кэш инструкций, загружаемый блочными пересылками (burst access)
- кэш контроллер (у i80960KB)
- 16 глобальных и 16 локальных регистров, кэш на 4 набора локальных регистров
- register scoreboarding (см. с. 62)

i80960Sx (1988, 16 MIPS):

- 16-бит внешняя шина данных
- Гарвардская архитектура: *512-байт* кэш инструкций + *256-байт* кэш данных
- встроенное FPU производительностью 0.5 MFlops (у i80960SB)

180960Сх (600000 транзисторов, 33 MHz, бб MIPS):

- 32-бит шины адреса и данных
- 128 битные внутренние шины данных
- *2* инструкции за цикл
- 1Кб кэш инструкций
- встроенное 1Кб ОЗУ

- устройство предсказания переходов
- встроенная поддержка 4-х каналов DMA (Direct Memory Access)

i80960Jx (1994, 50 MHz, 45 MIPS):

- *1* инструкция за цикл
- *8* наборов локальных регистров
- встроенный контроллер прерываний, возможность управлять кэшированием для размещения таблицы прерываний, обработчиков прерываний и их стеков в кэшируемой области
- *4Кб* кэш инструкций + *2Кб* кэш данных (у i80960JF)
- встроенное 1Кб ОЗУ
- *2 встроенных таймера*

i809б0Hx (1995, 75 MHz, 150 MIPS):

- электрически и логически (т.е. по ножкам микросхемы и по исполняемому коду) совместим с i80960Cx
- *16Кб* кэш инструкций + *8Кб* кэш данных
- 32-бит шины адреса и данных с поддержкой конвейерных и блочных пересылок обеспечивают пропускную способность 160Мб. о
- встроенное *2Кб* ОЗУ
- встроенные 32-бит таймеры

i80960RP (1996, 33 MHz):

- 32-бит процессор, содержащий ядро i80960JF
- поддержка интерфейсов с двумя шинами PCI
- прямой доступ между шинами PCI и локальной шиной процессора
- встроенный контроллер памяти
- поддержка интерфейса шины 1гС
- энергопотребление меньше 3 Вт

Программная модель

Прикладной программе доступны следующие регистры.

32 целочисленных регистра общего назначения (РОН) r0 - r31:

содержат слово (32 бит). При этом

- пара **rN, r(N+1)** при четном **N** может содержать данные длиной 64 бит
- тройка **rN, r(N+1), r(N+2)** при **N,** кратном 4, может содержать данные длиной 96 бит
- четверка **rN, r(N+1), r(N+2), r(N+3)** при **N,** кратном 4, может содержать данные длиной 128 бит

Над такими операндами (длиной более 32 бит) определены только инструкции пересылки (load/store). Выделяют

g0 - gl5: это регистры **r0** – **r15** *(global* registers); являются общими для всех функций (т.е. не изменяются при вызове функций). Регистр **g15** зарезервирован в качестве указателя текущего стекового кадра (он используется процессором, когда надо сохранить или восстановить набор локальных регистров).

l0 - **l15:** это регистры **r16** - **r31** *(local* registers); не доступны ни функции, вызвавшей данную, ни функциям, вызванным данной. При вызове функции старый набор локальных регистров сохраняется во внутреннем кэше, и вызванная функция получает новый набор, в котором:

l0: содержит предыдущее значение указателя текущего стекового кадра

l1: содержит указатель стека

l2: содержит адрес возврата

l3 - l15: значения не определены

ip (instruction pointer register):

содержит адрес текущей инструкции. Доступен только по чтению и может быть использован при формировании адреса.

Целочисленные коды условия icc (integer condition code):

являются частью AC (Arithmetic Control Register), который не доступен пользовательской программе как регистр. Коды условия устанавливаются специальными инструкциями и используются в командах условного перехода.

Все режимы адресации памяти процессоров Intel 80960x можно записать одной формулой: адрес ячейки памяти есть

base + index * scale + displacement

где

- **base** - базовый регистр: r0 - r31, ip
- **index** - индексный регистр: r0 - r31 (отсутствует, если ***base*** есть **ip)**
- **scale** - целая константа 1, 2, 4, 8, 16
- **displacement** - неотрицательное смещение 12 или 32 бит (12 бит смещение можно использовать только в режиме base + displacement).

Любой из элементов адреса может отсутствовать (с одним исключением: если отсутствует **index,** то должен отсутствовать и **scale).**

Все инструкции (за исключением load/store) имеют операнды в регистрах и потому размер всех операндов равен размеру слова (32 бит). Подавляющее большинство инструкций - трехоперандные.

Отметим, что это единственный из рассматриваемых нами процессоров, в котором жестко (аппаратно) закреплено, что **стек растет вверх** (от старших адресов к младшим).

Процессоры ARM

Процессоры ARM, разрабатываемые фирмой ARM (Advansed RISC Machines), играют значительную роль на рынке встраиваемых систем, особенно на рынке миниатюрных систем, сочетающих высокие пиковые нагрузки с длительными периодами ожидания (например, мобильные телефоны).

Компания ARM (Advansed RISC Machines) была основана в ноябре 1990 года фирмами

- Acorn Computers (информационные технологии для образования, Великобритания)
- Apple Computers
- VLSI Technology

Основной целью компании является разработка микропроцессорных ядер и их лицензирование широкому кругу производителей. В силу малости процессорного ядра ARM (всего 35000 транзисторов в базовом ядре ARM7) оно идеально подходит для интеграции в специализированные микросхемы потребителей. ARM Design Service Group постоянно работает с партнерами, обеспечивая ARM экспертизу OEM потребителям, желающим иметь встроенные в микросхемы решения на основе ядер ARM.

В настоящее время следующие компании лицензировали ARM и производят микросхемы на его основе:

- VLSI Technology
- Texas Instruments (TI)
- Samsung Corporation
- NEC Corporation
- GEC Plessey Semiconductors (GPS)
- Cirrus Logic
- Digital Equipment Corporation
- Symbios Logic
- Sharp Corporation
- Asahi Kasai Microsystems (AKM)
- European Silicon Structures (ES2)
- Lucky Goldstar Corporation
- Intel Corporation
- IBM Corporation

На основе ARM ядра разработано более 30 микропроцессоров и специализированных микросхем. Они находят применение в сотовых телефонах, органайзерах, модемах, графических ускорителях, видеофонах, камерах, телефонных коммутаторах, игровых приставках, дисковых накопителях, высокопроизводительных рабочих станциях, автомобильных навигационных системах, цифровых декодерах, smart картах, лазерных принтерах.

Основные отличительные черты архитектуры ARM:

1. Все инструкции являются условными (т.е. выполняются, только если код условия совпадает с кодом, указанным в инструкции). Это позволяет увеличить плотность кода и уменьшить потребность в инструкциях близкого перехода. Как следствие, нет отдельных команд условного перехода.
2. Все целочисленные арифметические инструкции могут выполнять операцию сдвига над операндами за тот же цикл, что выполняется и сама инструкция. Как следствие, нет отдельных команд сдвига.
3. Нет целочисленной инструкции деления.
4. Возможность выполнять DSP-подобные функции:
   А. присутствуют инструкции умножения и умножения со сложением (multiply-accumulate (MLA))
   Б. присутствуют инструкции блочного чтения из памяти и блочной записи в память, позволяющие переслать любое подмножество из 16-ти регистров общего назначения.
5. Некоторые модели могут работать в так называемом THUMB режиме: инструкции кодируются 16-ю битами вместо 32-х. Это значительно увеличивает плотность кода, но накладывает ряд ограничений на систему команд:
   А. полноценно доступны только 8 регистров из 16-ти, остальные могут ограниченно использоваться только в некоторых инструкциях (например, MOV, ADD и CMP);
   Б. не поддерживается условное исполнение инструкций, как следствие, появилась новая инструкция условного перехода;
   В. не поддерживается операция сдвига над операндами в целочисленных арифметических инструкциях, как следствие, появились новые инструкции сдвига;
   Г. все инструкции двухоперандные (а не трехоперандные как в обычном режиме).

Основные члены семейства

Каждый процессор в приведенных ниже технических данных в-основном обладает всеми возможностями предыдущих процессоров для обеспечения совместимости.

ARM1: Прототип, использовался только в тестовых системах

ARM2 (8 MHz, 4.7 MIPS):

64Кб адресное пространство

ARM3 (33 MHz, 18 MIPS):

- ARM2 ядро
- 4Кб единый кэш
- интерфейс сопроцессора
- добавлена новая инструкция SWP для работы с семафорами

ARM6 (36000 транзисторов, 33 MHz, 28 MIPS):

- 4Гб адресное пространство
- bi-endian (big- и little-endian порядок байтов)

- интерфейс сопроцессора

ARM600:

ARM6 со встроенным MMU

ARM7 (35000 транзисторов):

- ARM6 ядро, способное работать на повышенной частоте
- 3-х стадийный конвейер
- улучшенная инструкция аппаратного умножения (нужна для работы DSP)

ARM7D:

ARM7 с поддержкой отладки

ARM7DM:

ARM7D с улучшенным умножением

ARM7DMI (40 MIPS):

ARM7DM с ICEbreaker (встроенная поддержка In Circuit Emulation)

ARM70DM:

ARM7DMI (как отдельная микросхема)

ARM700 (40 MHz, 36 MIPS):

- ARM7 ядро
- 4Кб единый кэш
- writeback buffer
- встроенное MMU

ARM7500:

- ARM7 ядро
- 8Кб единый кэш
- writeback buffer
- встроенное MMU
- встроенный IOMD
- встроенный видеопроцессор

ARM7Txx:

ARM7xx (xx - одно из приведенных выше сочетаний) с поддержкой **THUMB** режима

ARM8 (80 MHz, 80 MIPS):

- совместим с ARM6 и ARM7
- 5-ти стадийный конвейер
- спекулятивное исполнение

StrongARM (SA110: 100 MHz, 115 MIPS; 200 MHz, 230 MIPS):

- высокоскоростной вариант ARM ядра, разработан совместно ARM ltd и Digital
- 16Кб кэш инструкций + 16Кб кэш данных (Гарвардская архитектура)
- глубокий конвейер
- полная совместимость кода не гарантируется в силу появления глубокого конвейера и раздельного кэша

AMULET2e (40 MIPS):

- это асинхронная версия ARM6, более быстрая, чем ARM7, но более медленная, чем ARM8
- 150 mW в активном состоянии, 0.1 mW в состоянии ожидания
- малое потребление мощности и механизм использования энергии делают AMULET2e идеальным процессором для приложений, где периоды высокой вычислительной нагрузки сочетаются с длительными периодами ожидания ввода

Программная модель

Прикладной программе доступны

16 целочисленных регистров общего назначения (РОН) r0 - r15:

содержат слово (32 бит). Некоторые из этих регистров имеют специальное назначение:

r15 - program counter (pc):

содержит адрес инструкции, находящейся через две инструкции от исполняемой в данный момент (т.е. адрес текущей инструкции + 8; при записи в этот регистр происходит переход по записанному адресу

r14 - link register (lr):

после инструкции Branch and Link (BL) (вызов функции) содержит адрес следующей инструкции (адрес возврата); во всех остальных инструкциях это обычный РОН

Коды условия cc (condition code):

являются частью **CPSR** (Current Program Status Register), который не доступен пользовательской программе как регистр. Коды условия устанавливаются по результату арифметических операций (если это указано в инструкции) или специальными инструкциями и используются для определения того, нужно ли исполнять текущую инструкцию (напомним, все инструкции являются условными).

Поддерживаются следующие режимы адресации памяти.

Для чтения/записи слова (32 бит) или беззнакового байта:

имеется один режим адресации

базовый регистр ± смещение

где **базовый регистр** - один из РОН **r0** - **r15,** а **смещение** может иметь три вида:

- **константа** - 12-битная константа

- **регистр** - один из РОН **rO** - **r15**
- **(регистр <операция> константа)** где
  - **регистр** - один из РОН **r0** - **r15,**

- **<операция>** - одна из следующих операций сдвига

  LSL: Logical Shift Left, сдвиг влево, вдвигаются нули

  LSR: Logical Shift Right, сдвиг вправо, вдвигаются нули

  ASR: Arithmetic Shift Right, сдвиг вправо, вдвигается знаковый разряд операнда регистр

  ROR: ROtate Right, циклический сдвиг вправо

  RRX: ROtate Right with eXtend циклический сдвиг вправо 33-х битной

  величины (<бит переносах регистр) на 1, правый операнд (т.е. константа) должен отсутствовать

- **константа** - 5-бит константа, задающая число сдвигов

Каждый из трех видов режима адресации имеет три варианта (что дает девять режимов адресации):

- **обычный,** адрес есть сумма базового регистра и смещения
- **с преиндекированием,** адрес есть сумма базового регистра и смещения, если инструкция чтения/записи выполнена (удовлетворен ее код условия), то адрес записывается в базовый регистр
- **с постиндексированием,** адрес есть базовый регистр, если инструкция чтения/записи выполнена, то сумма базового регистра и смещения записывается в базовый регистр

Для чтения/записи полуслова (16 бит) или чтения знакового байта:

(есть только в архитектуре ARM4 и выше) имеется один режим адресации

базовый регистр ± константа

где **базовый регистр** - один из РОН r0 - r15, а константа - 8-битная константа. Этот режима адресации имеет три варианта: обычный, с преиндекированием и с постиндексированием.

Все инструкции имеют поле кода условия (4 бита). Если текущее состояние флагов в регистре CPSR совпадает с указанным в поле кода текущей инструкции, то она будет выполнена, иначе - пропущена.

Все инструкции (за исключением load/store) имеют операнды в регистрах и потому размер всех операндов равен размеру слова (32 бит).

Все арифметические инструкции (включая логические) имеют бит, при установке которого по результату операции будут выставлены коды условия.

ГЛАВА-V. ШИНЫ

ЛЕКЦИЯ 5.1. АРХИТЕКТУРА ШИННЫХ СИСТЕМ. АРХИТЕКТУРА ШИНЫ (VME, PCI).

План:

- Архитектура системной шины
- Шина VME
- Шина PCI

Архитектура системной шины

Системная шина обеспечивает взаимодействие процессора и периферийных устройств. Поскольку часто основной задачей системы реального времени является управление тем или иным оборудованием, то качеству системной шины для промышленных компьютеров уделяется повышенное внимание.

В настольных компьютерах и промышленных системах распространены следующие системные шины.

Сравнение системных шин	
Название шины	**Производительность Мб/с**
PC/XT (8 бит)	4.7
PC/AT (16 бит)	16.66
MULTIBUS 1	24
EISA	33
VME32	40
MCA32	33
MULTIBUS 2	70
VME64	80
NUBUS	80
PCI32	132
VLB32	135
MCA64	160
AUTOBAHN 1	200
PCI64	264
AUTOBAHN 2	400
PCI64-66	528
FUTUREBUS+	1000

Архитектура шины VME

Стандарт на шину VME появился в 1981г., когда фирмы Motorola, Most ek и Signetics договорились об едином стандарте на шину для промышленных систем. В основу электрической спецификации шины был положен стандарт на шину VERSAbus фирмы Motorola, а в основу механической спецификации - стандарт Eurocard консорциума европейских компаний. Новый стандарт был назван VERSAmodule Eurocard, сокращенно VME. Однако, все компании, кроме Motorola, отказались расшифровывать VME как “VERSAmodule Eurocard”, поскольку “VERSAmodule” является зарегистрированной торговой маркой Motorola. Поэтому в настоящее время считается, что термин “VME” является единым обозначением, а не аббревиатурой. На шину VME существует стандарт IEEE 1014-1987.

Плата шины VME может быть одинарной или двойной высоты. Плата одинарной высоты (или 3U) имеет размеры 100мм х 160мм и один 96 контактный разъем DIN 41612, называемый P1. Плата двойной высоты (или 6U) имеет размеры 233мм х 160мм и второй 96 контактный разъем DIN 41612, называемый P2. Лицевая сторона платы имеет ширину 20мм. Платы разного размера могут работать совместно в одной системе.

На шине VME может быть до 21 разъема. Питание на платы подается через разъемы шины.

Основные характеристики шины.

- VME является асинхронной шиной.
- VME использует идею полного отображения на память. Каждое устройство представляется как адрес памяти или группа адресов.
- VME имеет раздельные шины адреса и данных. Плата одинарной высоты имеет 24битную шину адреса и 16-ти битную шину данных, плата двойной высоты - 32-битные шины адреса и данных.
- Возможность блочных пересылок, когда адрес задается только один раз.
- Поддержка мультипроцессорности:
  - приоритетный доступ к разделяемым ресурсам
  - возможность параллельного или исключительного доступа к разделяемым ресурсам
  - 4 уровня доступа
  - структура доступа master /'slave
  - защита разделяемых ресурсов посредством неделимых циклов шины
- 7 уровней запроса прерываний

Типичный обмен по шине состоит из цикла арбитража (для получения доступа к шине), адресного цикла (для выборки устройства), цикла реального обмена данными.

Шина VME как бы состоит из 4-х подшин.

1. **Шина арбитража** отвечает за определение приоритета запроса и разделение шины. В качестве арбитра выступает плата, находящаяся в разъеме 1 шины VME.
2. **Шина обмена данными** включает
   - шину данных, по которой возможны обмены размером 8/16/32 бит
   - шину адреса, по которой возможны обмены размером 16/24/32 бит

- управляющую шину, по которой передаются размеры адреса и данных, тип обмена (чтение или запись), признак окончания обмена, признак ошибки и т.д.

3. **Шина управления прерываниями** отвечает за приоритетное управление прерываниями
4. **Служебная шина** отвечает за подачу напряжений питания +5В, -12В, +12В на платы, информирование последних о проблемах с питанием, подачу периодического сигнала, инициализацию и сброс шины.

Блочный режим позволяет передать до 256 байтов по шине данных, однократно задав адрес на шине адреса. Следовательно, шина адреса неиспользуется большую часть времени обмена. Стандарт VME64 определяет 64-битные передачи за счет не используемой шины адреса: младшая часть 64-битного блока передается по шине данных, а старшая часть - по простаивающей шине адреса.

Архитектура шины PCI

Шина PCI (Peripheral Components Interconnect) появилась как стандарт фирмы Intel на соединение локальной шины процессора (соединяющей его с памятью) и периферийных устройств. В 1992г. появилась версия 1 стандарта, а в 1993г. - версия 2. В 1995г. появился стандарт на расширение PCI-66MHz.

Основные характеристики шины.

- PCI является синхронной шиной, частота синхронизации 33MHz или 66 MHz.
- PCI использует три адресных пространства:
  1. пространство памяти
  2. пространство ввода / вывода
  3. пространство регистров конфигурации
- PCI имеет мультиплексированную шину адреса и данных размером 32 бит или 64 бит.
- Возможность подключения до 256 шин PCI
- Параллельная работа различных шин
- Буферизация обменов
- Протокол конфигурации устройств