3.7 Процессоры общего назначения. Архитектуры CISC и RISC. Конвейер. Суперскалярность. Кэширование команд и данных.

Процессор - устройство для автоматического выполнения последовательности операций (или команд), предусмотренных программой. Среди операций могут быть арифметические, логические, обмена информацией с другими устройствами и другие. Процессор связан с запоминающими и периферийными (внешними) устройствами с помощью шинного интерфейса. Процессор состоит из операционных (арифметико-логического или АЛУ), управляющего устройств, внутренней (процессорной или регистровой) памяти и устройств управления вводом-выводом команд и данных. В процессе развития архитектура процессора усложняется, в его состав могут включаться и другие блоки.

Операционное устройство выполняет преобразование информации, представленной в двоичном виде. Преобразуемая информация называется операнд, команда преобразования - оператор.

Устройство управления предназначено для управления ходом вычислений, определяет последовательность выполнения операций, управляет выборкой команд и данных из памяти, их расшифровкой, вырабатывает сигналы управления элементарными действиями.

Архитектура ЭВМ и процессора - совокупность характеристик программных и аппаратных средств, определяющая комплекс существенных для потребителя свойств. Выделяют несколько групп свойств:

1. характеристики внутреннего языка и внутреннего математического обеспечения, характеристики системы команд - количество и типы форматов команд, мощность и номенклатура системы операции и системы адресации, структура и назначение пользовательских регистров процессора, форматы данных; характеристики этой группы определяют алгоритмические возможности процессора;

2. технические и эксплуатационные характеристики - разрядность процессора и памяти, эффективное быстродействие, параметры надежности, габаритные размеры, потребляемая мощность, масса процессора;

3. характеристики функциональных модулей базовой и расширенной конфигурации процессора и ЭВМ - арифметический сопроцессор, канал прямого доступа к памяти, расширитель ввода-вывода, сверхоперативная память

4. характеристики интерфейса и системы прерывания - число причин и приоритетных уровней, количество и типы служебных регистров обслуживающих систему прерывания, время реакции на прерывание.

Структура процессора - совокупность его функциональных блоков и связей между ними. Первоначальная "классическая" структура ЭВМ (фон Неймана) имела существенный недостаток - все операции ввода-вывода выполнялись через операционное устройство, на время выполнения этих операций другие работы прекращались [Справочник по цифровой выч.технике, с.192-]

Эволюция структуры и архитектуры ЭВМ и процессоров шла в направлении повышения производительности процессора. Вводились устройства, ориентированные на выполнение часто используемых операций, одновременно процессор освобождался от вспомогательных функций. Так возникли прямой доступ к памяти (ПДП), специализированные процессор ввода-вывода и многое другое.

Дальнейшее развитие идет в направлении создания многопроцессорных систем, к реализации элементов параллельности или конвейеров в отдельных процессорах. Кроме того, появилась и углубляется тенденция аппаратной реализации значительной части математического обеспечения, что проявляется в увеличении количества команд процессора - развитие архитектур CISC (Complex Instruction Set Computer - компьютер со сложной системой команд). С другой стороны, для упрощения разработки и быстродействия процессоров существует направление сокращения количества команд процессора - развитие архитектур RISC (Reduced Instruction Set Computer - компьютер с сокращенным набором команд).

Классификации процессоров:

1. по назначению:

- универсальные
- специализированные

2. по элементно-конструктивной базе:

- процессоры разных поколений
- однокристальные и многокристальные

3. по способу обработки и передачи информации:

- параллельного действия - обработка и передача информации производится словами, разрядность которых равна или пропорциональна разрядности процессора, таковы большинство современных процессоров
- последовательного действия

4. по режиму выполнения программ:

- однопрограммные (однозадачные)
- многопрограммные (многозадачные), в которых обеспечивается параллельная работа некоторых функциональных блоков

5. по сложности системы команд:

- CISC (Complex Instruction Set Computer)
- RISC (Reduced Instruction Set Computer)

6. по сложности системы адресации

7. по длине слова (4, 8, 16, 32 и более бит)

В настоящее время широко используются IBM-совместимые компьютеры с процессорами производства фирмы Intel или их аналоги, выпускаемые или выпускавшиеся фирмами AMD (Advanced Micro Devices), IBM, Cyrix, NexGen.

Регистры

Помимо ячеек оперативной памяти для кратковременного хранения данных можно использовать и регистры - ячейки, входящие в состав процессора и доступные из машинной программы. Доступ к регистрам осуществляется значительно быстрее, чем к ячейкам памяти, поэтому использование регистров заметно уменьшает время выполнения программ.

В современных процессорах Intel содержатся две группы регистров - прикладного и системного назначения. В данном пособии подробнее рассматриваются прикладные регистры или регистры прикладного программиста. Системные регистры и флаги только обозначены.

RISC (англ. *Reduced Instruction Set Computing*) — вычисления с сокращённым набором команд.

Это философия проектирования процессоров, которая во главу ставит следующий принцип: более компактные и простые инструкции выполняются быстрее. Простая архитектура позволяет как удешевить процессор, так и поднять тактовую частоту. Многие ранние RISC-процессоры даже не имели команд умножения и деления.

Идея создания RISC процессоров пришла после того как в 1970-х годах ученые из IBM обнаружили, что многие из функциональных особенностей традиционных ЦПУ игнорировались программистами. Отчасти это был побочный эффект сложности компиляторов. В то время компиляторы могли использовать лишь часть из набора команд процессора. Следующее открытие заключалось в том, что, поскольку некоторые сложные операции использовались редко, они как правило были медленнее, чем те же действия, выполняемые набором простых команд. Это происходило из-за того что создатели процессоров тратили гораздо меньше времени на улучшение сложных команд, чем на улучшение простых.

Первые RISC-процессоры были разработаны в начале 1980-х годов в Стэнфордском и Калифорнийском университетах США. Они выполняли небольшой (50 − 100) набор команд, тогда как обычные CISC (Complex Instruction Set computer) выполняли 100—200.

Характерные особенности RISC-процессоров:

- Фиксированная длина машинных инструкций (например, 32 бита) и простой формат команды.
- Одна инструкция выполняет только одну операцию с памятью — чтение или запись. Операции вида «прочитать-изменить-записать» отсутствуют.
- Большое количество регистров общего назначения (32 и более).

В настоящее время многие архитектуры процессоров являются RISC-подобными, к примеру, ARM, DEC Alpha, SPARC, AVR, MIPS, POWER и PowerPC. Наиболее широко используемые в настольных компьютерах процессоры архитектуры x86 ранее являлись CISC-процессорами, однако новые процессоры, начиная с Intel486DX, являются CISC-процессорами с RISC-ядром. Они непосредственно перед исполнением преобразуют CISC-инструкции процессоров x86 в более простой набор внутренних инструкций RISC.

CISC (англ. *Complex Instruction Set Computing*) — философия проектирования процессоров, которая характеризуется следующим набором свойств:

- Нефиксированным значением длины команды.
- Исполнение операций, таких как загрузка в память, арифметические действия кодируется в одной инструкции.

- Небольшим числом регистров, каждый из которых выполняет строго определённую функцию.

Типичными представителями являются процессоры на основе x86 команд (исключая современные Intel Pentium 4, Pentium D, Core, AMD Athlon, Phenom которые являются гибридными).

Наиболее распространённая архитектура современных настольных, серверных и мобильных процессоров построена по архитектуре Intel x86 (или x86-64 в случае 64-разрядных процессоров). Формально, все x86-процессоры являлись CISC-процессорами, однако новые процессоры, начиная с Intel486DX, являются CISC-процессорами с RISC-ядром. Они непосредственно перед исполнением преобразуют CISC-инструкции процессоров x86 в более простой набор внутренних инструкций RISC.
В микропроцессор встраивается аппаратный транслятор, превращающий команды x86 в команды внутреннего RISC-процессора. При этом одна команда x86 может порождать несколько RISC-команд(в случае процессоров типа P6-до 4-х RISC комманд в большинстве случаев). Исполнение команд происходит на суперскалярном конвейере одновременно по несколько штук.
Это потребовалось для увеличения скорости обработки CISC-команд, так как известно, что любой CISC-процессор уступает RISC-процессорам по количеству выполняемых операций в секунду. В итоге, такой подход и позволил поднять производительность CPU.

Конвейерная архитектура

Конвейерная архитектура (*pipelining*) была введена в центральный процессор с целью повышения быстродействия. Обычно для выполнения каждой команды требуется осуществить некоторое количество однотипных операций, например: выборка команды из ОЗУ, дешифрация команды, адресация операнда в ОЗУ, выборка операнда из ОЗУ, выполнение команды, запись результата в ОЗУ. Каждую из этих операций сопоставляют одной ступени конвейера. Например, конвейер микропроцессора с архитектурой MIPS-I содержит четыре стадии:

- получение и декодирование инструкции (Fetch)
- выполнение арифметических операций (Arithmetic Operation)
- адресация и выборка операнда из ОЗУ (Memory access)
- сохранение результата операции (Store)

После освобождения *k*-й ступени конвейера она сразу приступает к работе над следующей командой. Если предположить, что каждая ступень конвейера тратит единицу времени на свою работу, то выполнение команды на конвейере длиной в *n* ступеней займёт *n* единиц времени, однако в самом оптимистичном случае результат выполнения каждой следующей команды будет получаться через каждую единицу времени.

Действительно, при отсутствии конвейера выполнение команды займёт *n* единиц времени (так как для выполнения команды по прежнему необходимо выполнять выборку, дешифрацию и т. д.), и для исполнения *m* команд понадобится *n*m* единиц времени; при использовании конвейера (в самом оптимистичном случае) для выполнения *m* команд понадобится всего лишь *n* + *m* единиц времени.

Факторы, снижающие эффективность конвейера:

1. простой конвейера, когда некоторые ступени не используются (напр., адресация и выборка операнда из ОЗУ не нужны, если команда работает с регистрами);

2. ожидание: если следующая команда использует результат предыдущей, то последняя не может начать выполняться до выполнения первой (это преодолевается при использовании внеочередного выполнения команд, out-of-order execution);
3. очистка конвейера при попадании в него команды перехода (эту проблему удаётся сгладить, используя предсказание переходов).

Некоторые современные процессоры имеют более 30 ступеней в конвейере, что увеличивает производительность процессора, однако приводит к большому времени простоя (например, в случае ошибки в предсказании условного перехода.)

Принцип конвейера означает, что если мы сумеем разбить выполнение машинной инструкции на несколько этапов, то тактовая частота (а вернее, скорость, с которой процессор забирает данные на исполнение и выдает результаты) будет обратно пропорциональна времени выполнения самого медленного этапа. Если это время удастся сделать достаточно малым (а чем больше этапов на конвейере, тем они короче), то мы сумеем резко повысить тактовую частоту, а значит, и производительность процессора.

Процедуру выполнения практически любой инструкции можно разбить как минимум на пять непересекающихся этапов:

- Выборка инструкции (FETCH) из памяти. Из программы извлекается инструкция, которую нужно выполнить.
- Декодирование инструкции (DECODE). Процессор "соображает", что от него хотят, и переправляет запрос на нужное исполнительное устройство.
- Подготовка исходных данных для выполнения инструкции.
- Собственно выполнение инструкции (EXECUTE).
- Сохранение полученных результатов.

Конвейеризация потенциально применима к любой процессорной архитектуре, независимо от набора команд и положенных в ее основу принципов. Даже самый первый x86-процессор, Intel 8086, уже содержал своеобразный примитивный "двухстадийный конвейер" - выборка новых инструкций (FETCH) и их исполнение осуществлялись в нем независимо друг от друга. Однако реализовать что-то более сложное для CISC-процессоров оказалось трудно: декодирование неоднородных CISC-инструкций и их очень сильно различающаяся сложность привели к тому, что конвейер получается чересчур замысловатым, катастрофически усложняя процессор. Подобных трудностей у RISC-архитектуры гораздо меньше (а SPARC и MIPS, например, и вовсе были специально оптимизированы для конвейеризации), так что конвейеризированные RISC-процессоры появились на рынке много раньше, чем аналогичные x86.

Недостатки конвейера неочевидны, но, как обычно и бывает, из-за нескольких "мелочей" реализовать грамотно организованный конвейер совсем не просто. Основных проблем три.

- Необходимость наличия блокировок конвейера. Дело в том, что время исполнения большинства инструкций может очень сильно варьироваться. Скажем, умножение (и тем более деление) чисел требуют (на стадии EXECUTE) нескольких тактов, а сложение или побитовые операции - одного такта; а для операций Load и Store, которые могут обращаться к разным уровням кэш-памяти или к оперативной памяти, это время вообще не определено (и может достигать сотен тактов). Соответственно, должен быть какой-то механизм, который бы "притормаживал" выборку и декодирование новых инструкций до тех пор, пока не будут завершены старые. Методов решения этой проблемы много, но их развитие приводит к одному - в процессорах прямо перед исполнительными устройствами появляются специальные блоки-диспетчеры (dispatcher), которые накапливают

подготовленные к исполнению инструкции, отслеживают выполнение ранее запущенных инструкций и по мере освобождения исполнительных устройств отправляют на них новые инструкции. Даже если исполнение займет много тактов - внутренняя очередь диспетчера позволит в большинстве случаев не останавливать подготавливающий все новые и новые инструкции конвейер[Новые инструкции тоже не каждый такт удается декодировать, так что возможна и обратная ситуация: новых инструкций за такт не появилось, и диспетчер отправляет инструкции на выполнение "из старых запасов"]. Так в процессоре возникает разделение на две независимо работающие подсистемы: Front-end (блоки, занимающиеся декодированием инструкций и их подготовкой к исполнению) и Back-end (блоки, собственно исполняющие инструкции).

- Необходимость наличия системы сброса процессора. Поскольку операции FETCH и EXECUTE всегда выделены в отдельные стадии конвейера, то в тех случаях, когда в программном коде происходит разветвление (условный переход), зачастую оказывается, что по какой из веток пойти - пока неизвестно: инструкция, вычисляющая код условия, еще не выполнена[Вот здесь-то и нужны те самые режимы выставления флагов PowerPC, о которых шла речь в сноске 2]. В результате процессор вынужден либо приостанавливать выборку новых инструкций до тех пор, пока не будет вычислен код условия (а это может занять очень много времени и в типичном цикле страшно затормозит процессор), либо, руководствуясь соображениями блока предсказания переходов, "угадывать", какой из переходов скорее всего окажется правильным.
- Наконец, конвейер обычно требует наличия специального планировщика (scheduler), призванного решать конфликты по данным. Если в программе идет зависимая цепочка инструкций (когда инструкция-2, следующая за инструкцией-1, использует для своих вычислений данные, только что вычисленные инструкцией-1), а время исполнения одной инструкции (от момента запуска на стадию EXECUTE и до записи полученных результатов в регистры) превосходит один такт, то мы вынуждены придержать выполнение очередной инструкции до тех пор, пока не будет полностью выполнена ее предшественница. К примеру, если мы вычисляем выражение вида A•B+C с сохранением результата в переменной X (XfA•B+C), то процессор, выполняя соответствующую выражению цепочку из двух команд типа R4fR1•R2; R0fR3+R4, должен вначале дождаться, пока первая инструкция сохранит результат умножения A•B, и только потом прибавлять к полученному результату число C. Цепочки зависимых инструкций в программах - скорее правило, нежели исключение, а исполнение команды с записью результата в регистры за один такт - наоборот, скорее исключение, нежели правило, поэтому в той или иной степени с проблемой зависимости по данным любая конвейерная архитектура обязательно сталкивается. Оттого-то в конвейере и появляются сложные декодеры, заранее выявляющие эти зависимости, и планировщики, которые запускают инструкции на исполнение, выдерживая паузу между запуском главной инструкции и зависимой от нее.

Суперскалярность

У полноценной конвейеризации, более или менее эффективно обходящей перечисленные выше проблемы, есть одно несомненное достоинство: она настолько сложна, что, единожды реализованная, позволяет легко построить на ее основе целый ряд интересных новшеств. Для начала заметим, что коль уж у нас есть очереди готовых к исполнению инструкций и мы знаем взаимозависимости между ними по данным, есть техника переименования регистров, позволяющая разным инструкциям одновременно задействовать одни и те же регистры для разных целей, и, наконец, есть надежно работающая система сброса конвейера, то мы можем:

- Запускать на исполняющие устройства сразу несколько инструкций (если они не зависят друг от друга и могут быть безболезненно выполнены одновременно).
- Переупорядочивать независящие друг от друга инструкции так, как сочтем нужным.

Процессоры, использующие первую технику, называются суперскалярными. К примеру, сугубо теоретически, по числу исполнительных устройств, Pentium 4 может выполнять семь инструкций за такт, а Athlon 64 - девять. Реальные цифры, конечно, гораздо скромнее и определяются трудностью полноценной загрузки всех исполнительных устройств, однако Pentium 4 все же способен исполнять в устоявшемся режиме две (при некоторых условиях - четыре), а Athlon 64 - три инструкции за такт, одновременно производя две (A64 - три) операции по адресации и выборке данных из оперативной памяти. Может показаться, что реализация суперскалярного процессора очень проста (достаточно со стадии schedule просто распределять инструкции по разным исполнительным устройствам), однако такой лобовой подход обычно упирается в то, что Front-end процессора перестает успевать загружать исполнительные блоки работой. Поэтому на практике хорошо сделанные суперскалярные архитектуры, подобные AMD K7/K8, приходится специально "затачивать" под суперскалярность.

Процессоры, использующие вторую технику, называются процессорами с внеочередным исполнением инструкций (Out-of-Order processors, OoO). Техника переупорядочивания инструкций замечательна тем, что резко ослабляет негативные эффекты от медленной оперативной памяти и от наличия зависимых цепочек инструкций. Если, например, инструкция A обратилась к оперативной памяти, а нужных данных в кэше не оказалось или если A занимается ожиданием результатов выполнения какой-то другой инструкции, то OoO-процессор сможет пропустить вперед другие инструкции, не зависящие от результатов выполнения инструкции A. Кроме того, продвинутый планировщик OoO-процессора иногда может использоваться для реализации специфических деталей той или иной архитектуры - например, для спекулятивного исполнения по данным в случае Pentium 4 или одновременного исполнения нескольких веток программного кода в IA-64. Реализация OoO-процессоров не требует специальной оптимизации всего конвейера - это всего лишь усложнение схемы планировщиков, запускающих готовые к исполнению инструкции на исполнительные устройства в другом порядке, нежели они на планировщики поступили, плюс усложнение схем сброса конвейера и сохранения полученных результатов: результат выполнения прошедших вне очереди инструкций все равно должен сохраняться в последовательности, строго соответствующей расположению инструкций в изначальном коде[Это связано с тем, что если случится какая-то ошибка, то результаты выполнения запущенных вперед очереди инструкции придется аннулировать].

На сегодняшний день не существует ни одного суперскалярного или OoO CISC-процессора. Дело в том, что поскольку для нормальной реализации навороченных диспетчеров и планировщиков все равно требуется длительная и тщательная подготовка инструкций, причем желательно - до такого простого состояния, чтобы эти функционирующие на огромных частотах модули особенно не "задумывались" над тем, что такая хитрая последовательность байтов означает и куда ее следует направить (проблем у них и без того хватает), то любой исходный машинный код Front-end процессоров превращает перед исполнением в некое внутреннее, упрощенное и "разжеванное", состояние. То есть на этом этапе развития различия между RISC- и CISC-архитектурами почти стираются - просто у RISC’ов декодер, превращающий исходный машинный код в содержимое очередей планировщиков, устроен гораздо проще, чем "расковыривающий" хитро упакованные x86-инструкции CISC-подобный декодер AMD Athlon и Intel Pentium. Так что можно сказать, что фактически все современные x86-процессоры "в глубине души" являются полноценными RISC’ами - ведь исходный x86-код они в любом случае преобразуют на лету во внутреннее RISC-подобное представление. Правда, разной сложностью декодеров дело не ограничивается: все-таки классический RISC-код не только проще преобразовывать, но и результирующий внутренний код из него получается лучше - планировщикам гораздо легче его обрабатывать (в нем меньше зависимостей и операций с оперативной памятью).

Отличительной особенностью процессоров построенных по суперскалярной архитектуре является развитая схема управления потоками команд и данных, передаваемых в различные ФИУ (функциональные исполнительные устройства). Устройство предварительной выборки и диспетчеризации команд обеспечивает выборку команд в буфер команд, окончательную их дешифрацию, группировку и распределение для параллельного выполнения в конвейерных ФИУ. Буфер команд позволяет согласовать скорость работы памяти со скоростью обработки исполнительных устройств процессора. Команды могут быть предварительно выбраны из любого уровня иерархии памяти, например, из кэш-памяти команд, внешней кэш-памяти или основной памяти системы.

Все суперскалярные процессоры используют раздельное кэширование команд и данных.

Второй характерной чертой всех суперскалярных процессоров является использование методов предсказания переходов (что существенно влияет на эффективность суперскалярного процессора). Такими методами являются статический и динамический. Статический метод характерен тем, что для большинства условных переходов переход «назад» предсказывается совершающимся, а «вперед» - нет. Динамический основан на таблице истории переходов, обеспечивающий ускоренную обработку команд условного перехода. Поскольку направление перехода может меняться каждый раз, когда обрабатывается соответствующая команда, состояние прогноза должно каждый раз модифицироваться для отражения реального исхода перехода. Такая схема особенно эффективна при обработке циклов.

Кэширование

Кэширование — это использование дополнительной быстродействующей памяти (кэш-памяти) для хранения копий блоков информации из основной (оперативной) памяти, вероятность обращения к которым в ближайшее время велика.

Различают кэши 1-, 2- и 3-го уровней. Кэш 1-го уровня имеет наименьшую латентность (время доступа) но малый размер, кроме того кэши первого уровня часто делаются многопортовыми. Так процессоры AMD K8 умели производить 64 бит запись+64 бит чтение либо два 64-бит чтения за такт, процессоры Intel Core могут производить 128 бит запись+128 бит чтение за такт. Кэш 2-го уровня обычно имеет значительно большие латентности доступа, но его можно сделать значительно больше по размеру. Кэш 3-го уровня самый большой по объёму и довольно медленный, но всё же он гораздо быстрее, чем оперативная память.

На самом деле имеется в виду слово cache, что в переводе с французского означает "тайник, укромное место". Кэш-память большинства компьютеров разделяется на два уровня. Кэш-память первого уровня (Level 1 - L1) представляла собой память типа SRAM, интегрированную в микросхему процессора. Работал кэш L1 на частоте процессора. Кэш второго уровня первоначально располагался на системной плате и, естественно, мог работать только на частоте системной платы. Причем наличие кэша L2 считалось необязательным, и его нужно было докупать отдельно.

Когда кэш L1 полностью обновляется, чтение происходит из памяти, то есть в этот момент процессор все же вынужден перейти в состояние ожидания. Однако для уменьшения этого ожидания между ОП и кэшем L1 можно разместить еще один кэш (второго уровня - L2). Технология тут точно такая же. Скорость работы операций выборки из кэша L2 в старых процессорах была ниже, чем для кэша L1, но, во всяком случае, быстрее, чем у микросхем памяти.

Кэш L2, как было сказано, размещался на системной плате. Однако, поскольку разработчикам постоянно хотелось его убыстрить (вернее ускорить операции обмена данными между кэшем L1 и L2), в процессоре Pentium II он переместился на плату процессора (сама плата паковалась в специальный кожух, отчего корпус процессора имел несколько странный прямоугольный вид и монтировался на системной плате как какая-нибудь звуковая карта).

Уже начиная с процессора Pentium III и Athlon, кэш L2 стал частью микросхемы процессора и мог работать как на половинной, так и на полной частоте процессора.
Ситуация, когда процессор не находит нужные данные в кэше, называется промахом кэша. Если промах произошел в кэше L1, происходит обращение в кэш L2, соответственно, если промахнулся и второй, то приходится обращаться к оперативной памяти. Для процессоров Intel Itanium предусмотрена еще и кэш-память третьего уровня. Поскольку современные процессоры и оптимизирующие алгоритмы обеспечивают уровень промахов кэша не более 10%, то при операциях чтения обращение к оперативной памяти происходит довольно редко (не более чем в 1% случаев для двух кэшей и, соответственно, в 0,1% для трех кэшей). Поэтому, кстати, разгон оперативной памяти не очень ощутим. Разумеется, промахи кэша существуют только при операциях чтения. Любая команда изменения данных тут же приводит не только к изменению содержимого кэшей, но и соответствующей ячейки ОП.

При этом в современных процессорах кэш давно не является единым массивом памяти, как раньше, а разделен на несколько уровней. Наиболее быстрый, но относительно небольшой по объему кэш первого уровня (обозначаемый как L1), с которым работает ядро процессора, чаще всего делится на две половины - кэш инструкций и кэш данных. С кэшем L1 взаимодействует кэш второго уровня - L2, который, как правило, гораздо больше по объему и является смешанным, без разделения на кэш команд и кэш данных. Некоторые десктопные процессоры, по примеру серверных процессоров, также порой обзаводятся кэшем третьего уровня L3. Кэш L3 обычно еще больше по размеру, хотя и несколько медленнее, чем L2 (за счет того, что шина между L2 и L3 более узкая, чем шина между L1 и L2), однако его скорость, в любом случае, несоизмеримо выше, чем скорость системной памяти.

Кэш бывает двух типов: эксклюзивный и не эксклюзивный кэш. В первом случае информация в кэшах всех уровней четко разграничена - в каждом из них содержится исключительно оригинальная, тогда как в случае не эксклюзивного кэша информация может дублироваться на всех уровнях кэширования. Сегодня трудно сказать, какая из этих двух схем более правильная - и в той, и в другой имеются как минусы, так и плюсы. Эксклюзивная схема кэширования используется в процессорах AMD, тогда как не эксклюзивная - в процессорах Intel.

Отдельно стоит выделить I-кэш предназначенный для кэширование инструкций. Подмножество кода программы, наиболее «активно» исполняемое в данный отрезок времени, размещено в кэше инструкций (I-кэше). В зависимости от организации процессора, инструкции в этом кэше могут храниться в исходном неизменённом виде, либо в частично «предекодированном» виде, либо в полностью «декодированном» виде (то есть в виде готовых МОПов). В случае отсутствия в данном кэше нужных инструкций (исходных либо преобразованных) они считываются из кэша 2-го уровня (L2-кэша), при необходимости подвергаясь предварительному декодированию перед помещением в I-кэш.

Инструкции считываются из I-кэша блоками, с опережающей предвыборкой. Текущий блок инструкций отправляется сразу в два устройства — декодер инструкций и предсказатель переходов. Декодер преобразует исходные (либо частично декодированные) инструкции в микрооперации (МОПы), а предсказатель переходов определяет, есть ли в обрабатываемом блоке инструкции перехода, будут ли эти переходы совершены и по каким адресам. Если какой-либо

переход предсказывается как «совершённый», то немедленно считывается блок инструкций, находящийся по предсказанному адресу.

Например, в двуядерный процессор Conroe с 266-МГц FSB (FSB1066 с учетверённой скоростью передачи). Каждое ядро оснащено кэшем L1 на 64 кбайт, который разделён на кэш данных и кэш инструкций (по 32 кбайт каждый). Процессор использует впечатляющие 4 Мбайт кэша L2, который общий для двух ядер.