ОСНОВЫ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

БАГ
компьютерная академия

Урок №2

Центральный процессор

Содержание

Процессоры

Для начала давайте определимся, что такое процессор и для чего он нам необходим. Основной задачей процессора является выполнение команд и обработка данных, и команды и данные процессор получает из оперативной памяти, туда же он отправляет результаты своей работы

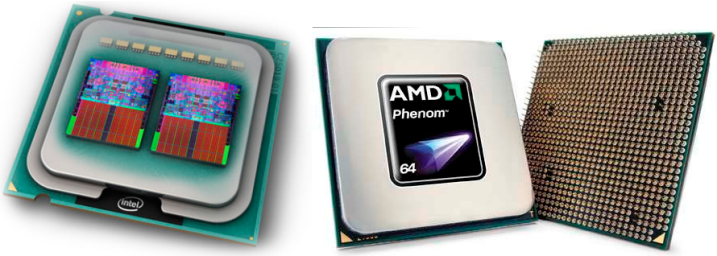

Рисунок 1

Процессор является основным вычислительным блоком компьютера, в наибольшей степенью определяющим его производительность.

Теперь мы переходим к рассмотрению основных характеристик и структурных элементов процессора, которые в первую очередь определяют его производительность.

Наборы команд процессора

Процессор — устройство, исполняющее команды, поэтому неудивительно, что первой и главной характеристикой процессора является тот набор команд, с которым

умеет работать данный процессор. Например, процессор вашего мобильного телефона скорее всего (бывают исключения) принципиально не может исполнять приложения, написанные для вашего PC. Сегодня существует целый ряд процессорных архитектур, однако в целом все процессоры можно условно разделить на две большие категории, в соответствии с фундаментальным подходом к формированию набора команд, который должен исполнять процессор.

RISC — *Reduced Instruction Set Computer* — процессор с сокращенной системой команд. Эти процессоры обычно имеют набор однородных регистров (внутренних ячеек памяти процессора, в которых хранятся обрабатываемые в данный момент данные) универсального назначения, причем их число может быть большим. Система команд отличается простотой, коды инструкций имеют четкую структуру, как правило фиксированной длины. В результате мы получаем аппаратную реализацию внутренней архитектуры, позволяющую с небольшими затратами декодировать и выполнять эти инструкции за минимальное (в пределе 1) число тактов.

CISC — *Complete Instruction Set Computer* — процессоры с полным набором инструкций, к которым относятся и семейство x86. Состав и назначение их регистров существенно неоднородны, широкий набор команд усложняет декодирование инструкций, на что расходуются аппаратные ресурсы. Возрастает число тактов, необходимых для выполнения инструкций. Преимуществом данной архитектуры является возможность решения очень большого количества разнородных задач.

Давайте рассмотрим на примере установки электрической лампочки, инструкции CISC.

Давайте рассмотрим на примере установки электрической лампочки, инструкции CISC.

- Возьмите электрическую лампочку.
- Вставьте ее в патрон.
- Вращайте до отказа.
- Конец.

И аналогичный пример в виде инструкций RISC.

- Поднесите руку к лампочке.
- Возьмите лампочку.
- Поднимите руку к патрону.
- Вставьте лампочку в патрон.
- Поверните ее.
- Лампочка поворачивается в патроне? Если да, то перейти к п. 5.
- Конец.

Многие инструкции RISC довольно просты, поэтому для выполнения какой либо операции потребуется больше таких инструкций. Их основное преимущество состоит в том, что процессор осуществляет меньше операций, а это, как правило, сокращает время выполнения отдельных команд и соответственно всей задачи (программы). В современных PC применяются CISC процессоры, с другой стороны в вашем телефоне или планшете наверняка установлен RISC процессор.

Теперь, рассмотрев различные подходы к формированию набора команд, мы можем охарактеризовать тот ключевой набор команд, который применяется в современных PC. Этот набор команд называется x86 и ориен-

тирован на вычисления с целыми числами. Также данный набор команд можем оперировать числами с плавающей точкой, но делает это настолько неэффективно, что уже на заре развития PC был разработан еще один набор команд (x87), который ориентирован на вычисления с плавающей точкой, все процессоры для PC очень давно поддерживают как минимум эти два набора команд. По-сути поддержка набора команд x86/x87 и позволяет отнести процессор к архитектуре PC. Помимо этих двух наборов команд, современные процессоры для PC поддерживают и другие, дополнительные наборы команд, а которых мы поговорим позже.

Тактовая частота

Быстродействие процессора во многом зависит от тактовой частоты, обычно сегодня измеряемую в гигагерцах (ГГц). Тактовая частота определяется параметрами кварцевого резонатора, представляющего собой кристалл кварца в оловянной оболочке. Под воздействие электрического напряжения в кристалле кварца возникают колебания электрического тока с частой, определяемой формой и размерами кристалла. Частота этого переменного тока и называется тактовой частотой. Наименьшей единицей времени для процессора, как для логического устройства является период тактовой частоты или просто такт. На каждую операцию (выполнение команды) процессор затрачивает некоторое количество тактов.

Естественно, чем выше тактовая частота процессора, тем производительнее он работает, так как в единицу времени происходит большее количество тактов и выпол-

няется большее количество команд. Естественно, более новые процессоры работают на все более высоких тактовых частотах (это достигается, в частности, улучшением технологии их изготовления) показываю большую производительность. Но тактовая частота не единственный фактор, определяющий производительность процессора. Ведь количество тактов, затрачиваемое на выполнение команд тоже можно менять. И если первые x86 процессоры на выполнение одной команды тратили в среднем около 12 тактов, в 286 и 386 этот показатель в среднем составлял около 4,5 тактов, в 486 — около 2 тактов, то в процессорах Pentium в среднем выполняется одна команда за такт. Ну а современные процессоры могут выполнять несколько команд одновременно (за счет параллельного исполнения команд). Различное количество тактов, затрачиваемое процессорами на выполнение команд, затрудняет их сравнение с использованием только лишь тактовой частоты. Гораздо удобнее использовать для измерения производительности среднее количество операций, выполняемое за единицу времени. Существует единица измерения этого параметра — MIPS — *million integer per second*, миллионы целочисленных операций в секунду, и MFLOPS — *million floating per second*, миллионы вещественных операций в секунду. Этими величинами измеряется производительность разных блоков процессора — ALU и FPU.

ALU традиционно отвечают за два типа операций: арифметические действия (сложение, вычитание, умножение, деление) с целыми числами, логические операции с опять-таки целыми числами (логическое «и», логиче-

ское «или», «исключающее или», и тому подобные). Что, собственно, и следует из их названия. Блоков ALU в современных процессорах, как правило, несколько.

FPU занимается выполнением команд, работающих с числами с плавающей запятой, Как и в случае с ALU, отдельных блоков в FPU может быть несколько, и они способны работать параллельно.

Как вы уже наверное догадались теоретически сравнивать два процессора нужно рассматривая быстродействие и тактовую частоту работы в совокупности: чем меньше тактов затрачивает в среднем процессор на исполнение команды, тем выше его эффективность (производительность) даже при неизменной тактовой частоте.

Например, 486 процессор (в среднем 2 такта на команду) на частоте 133 МГц работает даже медленнее, чем процессор Pentium (в среднем 1 такт на команду) на частоте 75МГц. Оценивать реальную производительность процессора в сравнении с другими весьма непросто, и нужно понимать, что такое сравнение во многом зависит от той задачи, которую процессор решает. Тактовые частоты современных процессоров типично лежат в диапазоне от 2 ГГц до 4 ГГц.

Шина данных процессора

Одной из самых общих характеристик процессора является разрядность его шины данных и шины адреса.

Когда говорят о шине процессора, обычно имеют ввиду шину данных, которая является набором соединений, для передачи и приема данных. Чем больше сигналов одновременно поступает на шину, тем больше данных

по ней передается за определенный интервал времени, и тем быстрее она работает. Разрядность шины данных подобна количеству полос автомагистрали — чем больше полос, тем больше поток машин, чем шире шина данных, тем больше данных за одинаковые промежутки времени по ней передается. В процессоре 286 для приема и передачи двоичных данных используется 16 соединений, поэтому их шина данных считается 16-разрядной. У процессоров с 32-х разрядной шиной данных (например, 486), таких соединений вдвое больше, поэтому за единицу времени они передают и получают вдвое больше данных, чем 16-и разрядные процессоры — разумеется, эффективность обмена данными у таких процессоров выше. Современные процессоры имеют 64-х разрядную шину данных, поэтому они могут передавать в системную память по 64 бита за один такт.

Таким образом мы определили, с помощью какой шины процессор связан с оперативной памятью и от разрядности этой шины конечно же зависит производительность процессора. Теперь давайте разберемся, как процессор обрабатывает полученные из оперативной памяти данные.

Регистры процессора

Хоть процессор и получает данные из оперативной памяти с помощью шины некоторой ширины, это не значит, что внутри он может обрабатывать данные такой же разрядности. Давайте разберемся, как это происходит.

Длина порции данных, которые может обработать процессор за один прием, характеризуется разрядностью внутренних регистров.

Регистр — это по существу ячейка памяти внутри процессора, например, процессор может складывать числа, записанные в двух разных регистрах, а результат записывать в третий регистр. Разрядность регистров описывает разрядность обрабатываемых процессором данных. Разрядность регистров определяет также характеристики программного обеспечения и команд, выполняемых процессором. Например, процессоры с 32-разрядными внутренними регистрами могут выполнять 32-разрядные команды, которые обрабатывают данные 32-разрядными порциями, а процессоры с 16-разрядными регистрами этого делать не могут. В современных процессорах для PC внутренние регистры являются 32 или 64 разрядными (доминируют в PC 64-х разрядные процессоры).

Шина адреса

Шина адреса представляет собой набор проводников, по которым контроллеру памяти передается адрес ячейки памяти, в которую или из которой пересылаются данные. По каждому проводнику передается один бит адреса, соответствующий одной цифре в адресе. Увеличение количества проводников (разрядов шины) используемых для формирования адреса, позволяет увеличить количество адресуемых ячеек. Разрядность шины адреса определяет максимальный объем памяти, адресуемой процессором. В компьютерах применяется, как Вы знаете, двоичная системы счисления. Если, например, разрядность шины адреса составила бы всего один бит (один провод для передачи данных), то по этому проводу можно было бы передать всего два значения (логический нуль — нет

напряжения, логическая единица — есть напряжение) и таким образом можно было бы адресоваться к двум ячейкам памяти. Использование двух бит для задания адреса позволило бы адресоваться уже к 4-м ячейкам памяти (00, 01, 10, 11 на шине — на четыре разных адреса можно указать). Вообще, количество разных значений, принимаемое n-разрядным двоичным числом равно 2 в степени n. Соответственно, при ширине шины адреса n бит, количество разных ячеек памяти, к которым можно адресоваться составляет 2 в степени n, поэтому говорят, что процессор поддерживает 2 в степени n байт оперативной памяти, или говорят, что адресное пространство процессора равно 2 в степени n байт. Например: процессор 8086 имел адресную шину 20 бит. Тогда он мог адресовать (2 в степени 20=1048576) байт оперативной памяти, т.е. 1 Мбайт. Таким образом, максимальный объем оперативной памяти, поддерживаемой процессором 8086 составляет 1 Мбайт. 286-ой процессор имел адресную шину равную 24 битам, адресуя таким образом уже 16 Мбайт (обратите внимание: каждый новый бит в шине адреса, увеличивает объем адресуемой памяти вдвое, это естественно, если вспомнить формулу *Объем памяти = 2 в степени разрядности шины*). Современные процессоры архитектуры 64х имеют адресную шину равную 40 бит, что соответствует максимальному объему поддерживаемой оперативной памяти — 1 Тбайт.

Шины данных и адреса независимы, и разработчики микросхем выбирают их разрядность по своему усмотрению. Разрядность этих шин является показателем возможностей процессора: разрядность шины данных определяет

возможности процессора быстро обмениваться информацией, разрядность адресной шины определяет объем поддерживаемой процессором памяти.

Частоты процессора и системной шины

Вы уже знаете, что современный процессор работает сегодня на частотах порядка 2–4 ГГц. С другой стороны оперативная память работает на сегодня на гораздо более низких частотах. Более того, не только оперативная память, но и чипсет работает на невысокой относительно процессора частоте. На материнской плате вообще нет компонентов, которые работали бы быстрее, чем системная шина. Дело в том, что чем больше физические размеры объектов, тем сложнее заставить их работать на большей частоте, микросхема процессора имеет очень небольшие физические размеры, в то время как материнская плата гораздо больше и не может работать на частоте несколько гигагерц. Давайте проведем мысленный эксперимент:

Пусть мы имеем компьютер, в котором установлен процессор с частотой 2 ГГц. Мы планируем заменить процессор на более производительный и установить новый процессор с частотой 3 ГГц.

Итак, вы извлекаете из процессорного гнезда 2 ГГц процессор и устанавливаете 3 ГГц. Вопрос: а материнская плата работает с новым процессором в прежнем режиме, в том же, в котором она работала и с процессором 2,0 ГГц? Если плата теперь работает в полтора раза быстрее, то спрашивается: если для того, чтобы ускорить процессор его нужно заменить, может быть чтобы уско-

рить во столько же раз материнскую плату ее, наверное, тоже нужно заменить на в полтора раза более производительную? Т.е. простой заменой процессора систему нельзя улучшить? Нужно заменять материнскую плату, на новой плате должен быть другой, способный работать на повышенной частоте чипсет, нужно заменить и оперативную память? Если бы все было так, как мы сейчас описали, то компьютер вообще бы практически не подлежал бы улучшению (*upgrade*), а можно было бы лишь заменить его практически целиком.

Но это не так. В большинстве систем предел скорости на которой работает чипсет, составляет около 800 МГц. Тогда возникает главный вопрос: каким образом в ОДНОЙ И ТОЙ ЖЕ материнской плате могут работать и 2 ГГц и 3 ГГц процессора? Если, к примеру, частота системной шины 800 МГц, как процессор работает на более высокой частоте? Ответ кроется в том, что процессор НЕ работает на частоте системной шины (на этой частоте работает вся материнская плата, чипсет, память, но не процессор).

Процессор лишь использует для своей работы частоту системной шины. Дело в том, что процессор УМНОЖАЕТ эту частоту на некоторый множитель, таким образом получая результирующую частоту, на которой и работает. Например: наш абстрактный процессор из мысленного эксперимента, работающий на частоте 2 ГГц использует системную шину на частоте 200 МГц, умножая ее на 10, а процессор, работающий на частоте 3 ГГц использует ту же частоту системной шины, умножая ее на 15.

Таким образом, в одну и ту же материнскую плату можно вставить различные процессоры, работающие

на разных частотах за счет того, что частота системной шины используется одна и та же, а процессор производит в себе умножение частоты поданной на него системной шины на некоторый, закрепленный в конкретном процессоре, множитель.

Еще один важнейший момент. Какая система быстрее? Процессор 2 ГГц = 400 МГц × 5 или 2 ГГц = 800 МГц × 2.5? Иными словами, влияет ли частота системной шины на производительность системы при неизменной частоте процессора? Естественно влияет. Разумеется, компьютер, у которого системная шина работает на частоте 800 МГц, при прочих равных условиях будет работать быстрее системы с тем же процессором, но на частоте системной шины 400 МГц.

Ведь и память, и прочие компоненты предпочтительнее использовать на более высокой частоте (естественно, если это позволяет их спецификация, если они, иными словами, для такой частоты предназначены). Т.е. компьютер производительнее не только потому, что в нем установлен более быстродействующий процессор; производительность так же зависит и от частоты системной шины, на которой, в свою очередь, работает оперативная память.

Важное замечание: естественно, производители чипсетов (а именно от чипсета зависит, какую частоту системной шины поддерживает материнская плата, от чипсета почти все зависит ☺) постоянно стремятся увеличить частоту системной шины, поддерживаемую их продуктами, так как даже при очень быстром процессоре система не может быстрее обмениваться данными с памятью, чем позволяет системная шина и это ограничивает производительность всей системы. И, изучая с Вами в ближайшее

время чипсеты, мы увидим, какое значение придается поддержке в сегодняшнем чипсете максимально возможной частоте системной шины и типу поддерживаемой памяти, так как, в конечном счете, борьба за быструю системную шины — это во многом борьба за ускорение обмена память — процессор.

С другой стороны, как было сказано выше, сегодня контроллер памяти установлен уже не на материнской плате (в чипсете), а размещен в самом процессоре, поэтому, на первый взгляд, материнской плате вообще не нужно ничего поддерживать для высокоскоростного обмена между процессором и памятью, но это только на первый взгляд. На самом деле память устанавливается не в процессор непосредственно, а в разъемы на материнской плате, поэтому производитель платы по прежнему должен обеспечивать высокоскоростной обмен между процессором и памятью. На самом деле все описанные выше рассуждения можно вообще не менять, а просто представить себе, что северный мост теперь просто стал частью процессора.

Теперь, рассмотрев основные характеристики и компоненты процессора, давайте обсудим прочие важные архитектурные элементы современных процессоров.

Декодер

На самом деле, исполнительные блоки всех современных десктопных x86-процессоров... вовсе не работают с кодом в стандарте x86. У каждого процессора есть своя, «внутренняя» система команд, не имеющая ничего общего с теми командами (тем самым «кодом»), которые

поступают извне. В общем случае, команды, исполняемые ядром — намного проще, «примитивнее», чем команды стандарта x86. Именно для того, чтобы процессор «внешне выглядел» как x86 CPU, и существует такой блок как декодер: он отвечает за преобразование «внешнего» x86-кода во «внутренние» команды, исполняемые ядром (при этом достаточно часто одна команда x86-кода преобразуется в несколько более простых «внутренних»). Декодер является очень важной частью современного процессора: от его быстродействия зависит то, насколько постоянным будет поток команд, поступающих на исполняющие блоки. Ведь они неспособны работать с кодом x86, поэтому то, будут они что-то делать, или простаивать — во многом зависит от скорости работы декодера.

Суперскалярная архитектура

Суперскалярная архитектура — Способность выполнения нескольких машинных инструкций за один такт процессора. Появление этой технологии привело к существенному увеличению производительности.

Основная черта всех современных процессоров состоит в том, что они способны запускать на исполнение не только ту команду, которую (согласно коду программы) следует исполнить в данный момент времени, но и другие, следующие после неё. Рассмотрим простой пример. Пусть нам следует исполнить следующую последовательность команд:

(1) A = B + C
(2) Z = X + Y
(3) K = A + Z

Легко заметить, что команды (1) и (2) совершенно независимы друг от друга — они не пересекаются ни по исходным данным (переменные B и C в первом случае, X и Y во втором), ни по месту размещения результата (переменная A в первом случае и Z во втором). Стало быть, если на данный момент у нас есть свободные исполняющие блоки в количестве более одного, данные команды можно распределить по ним, и выполнить одновременно, а не последовательно. Таким образом, если принять время исполнения каждой команды равным N тактов процессора, то в классическом случае исполнение всей последовательности заняло бы N*3 тактов, а в случае с параллельным исполнением — всего N*2 тактов (так как команду (3) нельзя выполнить, не дождавшись результата исполнения двух предыдущих).

> ***Замечание:*** *разумеется, степень параллелизма не бесконечна: команды могут быть выполнены параллельно только в том случае, когда на данный момент времени есть в наличии соответствующее количество свободных от работы блоков функционального устройства (ФУ), причём именно таких, которые «понимают» рассматриваемые команды. Самый простой пример: блок, относящийся к ALU, физически неспособен исполнить инструкцию, предназначенную для FPU.*

На самом деле всё ещё несколько сложнее. Пусть у нас имеется следующая последовательность:

(1) A = B + C
(2) K = A + M
(3) Z = X + Y

Теперь очередь исполнения команд процессором будет изменена: так как команды (1) и (3) независимы друг от друга (ни по исходным данным, ни по месту размещения результата), они могут быть выполнены параллельно — и будут выполнены параллельно. А вот команда (2) будет выполнена после них (третьей) — поскольку для того, чтобы результат вычислений был корректен, необходимо, чтобы перед этим была выполнена команда (1). Такой механизм называется «внеочередным исполнением команд» (*Out-of-Order Execution*, или сокращённо «OoO»): в тех случаях, когда очерёдность выполнения никак не может сказаться на результате, команды отправляются на исполнение не в той последовательности, в которой они располагаются в коде программы, а в той, которая позволяет достичь максимального быстродействия.

Теперь вам должно стать окончательно понятно, зачем современным CPU такое количество однотипных исполняющих блоков: они обеспечивают возможность параллельного выполнения нескольких команд, которые в случае с «классическим» подходом к проектированию процессора пришлось бы выполнять в той последовательности, в которой они содержатся в исходном коде, одну за другой.

Процессоры, оснащённые механизмом параллельного исполнения нескольких подряд идущих команд, принято называть «суперскалярными». Однако не все суперскалярные процессоры поддерживают внеочередное исполнение. Так, в первом примере нам достаточно «простой суперскалярности» (выполнения двух последовательных команд одновременно), а вот во втором примере без перестановки команд местами уже

не обойтись, если мы хотим получить максимальное быстродействие. Все современные CPU обладают обоими качествами: являются суперскалярными, и поддерживают внеочередное исполнение команд. В то же время, были в истории x86 и «простые суперскаляры», OoO не поддерживающие.

Конвейерная архитектура

Конвейерная архитектура (*pipelining*) была введена в центральный процессор с целью повышения быстродействия.

Рисунок 2

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. Давайте посмотрим как это происходит (рис. 2).

Современные процессоры обрабатывают данные конвейерным способом, т.е. одновременно в процессоре на различных стадиях происходит выполнение нескольких команд — одна команда практически близка к исполнению, другая — в середине процесса исполнения, третья только входит на конвейер и т.д., на рисунке пример первого из применяемых в PC конвейера. Таким образом удается значительно повысить производительность, так как все части процессора одновременно задействуются, и каждая часть процессора выполняет свой маленьких объем работы. При этом вводят понятие «количество стадий конвейера» — число элементарных задач на которые разбивается выполнение команды — на рисунке выделена каждая стадия. При этом если стадий мало, то недостаточно полно используются все части процессора одновременно (страдает производительность), и невозможно использовать большие тактовые частоты. А если стадий много, то появляется возможность использовать более высокие частоты процессора, но при этом увеличиваются задержки в процессе исполнения команд (тоже страдает производительность). Таким образом, производители процессоров выбирают некоторое оптимальное количе-

ство стадий конвейера, при котором процессор показывает наивысшую производительность. Количество стадий в конвейерах современных процессоров от 15 до 30, при этом производители столкнулись с ещё одной проблемой — хоть частоты процессоров одних производителей благодаря длинным конвейерам (30 стадий) выросли более 3-х гигагерц, но их производительность сравнима с процессорами конкурентов, у которых конвейер 15 стадий, и частоты намного меньше. Также, при работе конвейера возникает проблема, связанная с исполнением команд ветвления (условных переходов): если в процессор поступила команда ветвления, то ответ на вопрос о том, какие следующие команды должны быть исполнены будет известен ТОЛЬКО после того, как исполниться команда ветвления. Но пока она не исполнена, на конвейер могли бы поступить другие команды, которые бы постепенно исполнялись! Получается, что пока не будет выполнена команда ветвления, конвейер должен простаивать! И чем длиннее конвейер, тем дольше он простаивает.

Давайте подробнее обсудим, как современный процессор ведет себя в такой ситуации.

В любой более-менее сложной программе присутствуют команды условного перехода: «Если некое условие истинно — перейти к исполнению одного участка кода, если нет — другого». С точки зрения скорости выполнения кода программы современным процессором, поддерживающим внеочередное исполнение, любая команда условного перехода — большая проблема. Ведь до тех пор, пока не станет известно, какой участок кода после условного перехода окажется «актуальным» — его невоз-

можно начать декодировать и исполнять. Для того чтобы как-то примирить концепцию конвейера с командами условного перехода, предназначается специальный блок: блок предсказания ветвлений. Как понятно из его названия, занимается он, по сути, «пророчествами»: пытается предсказать, на какой участок кода укажет команда условного перехода, ещё до того, как она будет исполнена. В соответствии с указаниями «штатного внутриядерного пророка», процессором производятся вполне реальные действия: «напророченный» участок кода загружается и начинается декодирование и выполнение его команд. Причём, среди выполняемых команд также могут содержаться инструкции условного перехода, и их результаты тоже предсказываются, что порождает целую цепочку из пока не проверенных предсказаний! Разумеется, если блок предсказания ветвлений ошибся, вся проделанная в соответствии с его предсказаниями работа просто аннулируется и содержимое конвейера очищается.

На самом деле, алгоритмы, по которым работает блок предсказания ветвлений, вовсе не являются шедеврами искусственного интеллекта. Преимущественно они просты. Ибо чаще всего команда условного перехода встречается в циклах: некий счётчик принимает значение X, и после каждого прохождения цикла значение счётчика уменьшается на единицу. Соответственно, до тех пор, пока значение счётчика больше нуля — осуществляется переход на начало цикла, а после того, как он становится равным нулю — исполнение продолжается дальше. Блок предсказания ветвлений просто анализирует результат выполнения команды условного перехода, и считает, что

если N раз подряд результатом стал переход на определённый адрес — то и в N+1 случае будет осуществлён переход туда же. Однако, несмотря на весь примитивизм, данная схема работает просто замечательно: например, в случае, если счётчик принимает значение 100, а «порог срабатывания» предсказателя ветвлений (N) равен двум переходам подряд на один и тот же адрес — легко заметить, что 97 переходов из 98 будут предсказаны правильно!

Разумеется, несмотря на достаточно высокую эффективность простых алгоритмов, механизмы предсказания ветвлений в современных CPU всё равно постоянно совершенствуются и усложняются — но тут уже речь идёт о борьбе за единицы процентов: например, за то, чтобы повысить эффективность работы блока предсказания ветвлений с 95 процентов до 97, или даже с 97% до 99...

Предсказание ветвлений позволяет существенно ускорить выполнение программы, но при этом возникает проблема — в случае, если предсказание окажется неверным, всё содержимое конвейера тоже окажется неверным, и его придется очищать. А чем длиннее конвейер, тем больше будет потеря времени в результате его очистки, поэтому производители процессоров постоянно совершенствуют механизм предсказания ветвлений.

Упреждающее выполнение

Еще одним важным инструментом оптимизации исполнения команд процессором является упреждающее выполнение. Эта технология подразумевает начало исполнения инструкции до готовности всех операндов. При этом выполняются все возможные действия, и декоди-

рованная инструкция с одним операндом помещается в исполнительное устройство, где дожидается готовности второго операнда, выходящего с другого конвейера. С помощью этого метода процессор просматривает стоящие на очереди команды и выполняет те из них, к которым вероятно потребуется обратиться позже. Таким образом ряд команд процессор может выполнить заранее, а затем пользоваться результатами произведенных вычислений позже.

Блок предвыборки данных

Еще одна технология, позволяющая повысить производительность процессора — предварительная выборка данных (*Prefetch*). Задача данного блока — предварительно загружать данные, которые процессору, вероятно, скоро понадобятся. Этот инструмент очень похож по принципу своего действия на блок предсказания ветвлений, с той только разницей, что в данном случае речь идет не о коде, а о данных. Общий принцип действия такой же: если встроенная схема анализа доступа к данным в ОЗУ решает, что к некоему участку памяти, скоро будет осуществлен доступ, она даёт команду на загрузку данного участка памяти в специальную, очень быструю память, которая называется кеш (об этой памяти мы поговорим ниже) ещё до того, как он понадобится исполняемой программе.

«Умно» (результативно) работающий блок предвыборки позволяет существенно сократить время доступа к нужным данным, и, соответственно, повысить скорость исполнения программы. Также грамотный Prefetch очень хорошо компенсирует высокую латентность подсистемы

памяти, подгружая нужные данные в кеш, и тем самым, нивелируя задержки при доступе к ним, если бы они находились не в кеше, а в основном ОЗУ.

Однако, разумеется, в случае ошибки блока предвыборки данных, неизбежны негативные последствия: загружая де-факто «ненужные» данные в кеш, Prefetch вытесняет из него другие (быть может, как раз нужные). Кроме того, за счёт «предвосхищения» операции считывания, создаётся дополнительная нагрузка на контроллер памяти (де-факто, в случае ошибки — совершенно бесполезная).

Алгоритмы Prefetch, как и алгоритмы блока предсказания ветвлений, тоже не блещут интеллектуальностью: как правило, данный блок стремится отследить, не считывается ли информация из памяти с определённым «шагом» (по адресам), и на основании этого анализа пытается предсказать, с какого адреса будут считываться данные в процессе дальнейшей работы программы. Впрочем, как и в случае с блоком предсказания ветвлений, простота алгоритма вовсе не означает низкую эффективность: в среднем, блок предвыборки данных чаще «попадает», чем ошибается (и это, как и в предыдущем случае, прежде всего связано с тем, что «массированное» чтение данных из памяти, как правило происходит в процессе исполнения различных циклов).

Описанные выше технологии часто объединяют общим названием: динамическое исполнение команд.

Кеш память

Еще одним важным компонентом современного процессора, точнее подсистемы процессор-оперативная память является кеш память.

Давайте рассмотрим, как происходит обмен информацией между процессором и памятью. В большинстве случаев оперативная память не удовлетворяет потребностям современных процессоров в плане пропускной способности, т.к работает на значительно более низких частотах. Современный процессор работает на частотах около 3 ГГц и, естественно, что при обмене с памятью процессор достаточно долгое время будет ждать прихода новых порций данных и таким образом простаивать. Для того чтобы этого избежать, между памятью и процессором устанавливают дополнительно небольшой объем очень быстрой памяти, работающей без задержек на частоте процессора. Такая память и называется кеш-память, она имеет тип SRAM. В современном процессоре встроено некоторое количество такой памяти (от 32 Кб до 3 Мб) и эта память обеспечивает снижение простоев процессора при операциях с оперативной памятью.

В переводе слово кеш (*cache*) означает «тайный склад», «тайник» («заначка»). Тайна этого склада заключается в его «прозрачности» — для программы он не представляет собой дополнительной адресуемой области памяти. кеш является дополнительным быстродействующим хранилищем копий блоков информации из основной памяти, вероятность обращения к которым в ближайшее время велика. кеш не может хранить копию всей основной памяти, поскольку его объем во много раз меньше основной памяти.

Он хранит лишь ограниченное количество блоков данных и каталог (*cache directory*) — список их теку-

щего соответствия областям основной памяти. Кроме того, кешироваться может не вся память, доступная процессору.

При каждом обращении к памяти контроллер кеш-памяти по каталогу проверяет, есть ли действительная копия затребованных данных в кеше. Если она там есть, то это случай кеш-попадания (*cache hit*), и данные берутся из кеш-памяти. Если действительной копии там нет, это случай кеш-промаха (*cache miss*), и данные берутся из основной памяти. В соответствии с алгоритмом кеширования блок данных, считанный из основной памяти, при определенных условиях заместит один из блоков кеша. От интеллектуальности алгоритма замещения зависит процент попаданий и, следовательно, эффективность кеширования.

Поиск блока в списке должен производиться достаточно быстро, чтобы «задумчивостью» в принятии решения не свести на нет выигрыш от применения быстродействующей памяти. Обращение к основной памяти может начинаться одновременно с поиском в каталоге, а в случае попадания — прерываться (архитектура Look aside). Это экономит время, но лишние обращения к основной памяти ведут к увеличению энергопотребления. Другой вариант: обращение к внешней памяти начинается только после фиксации промаха (архитектура Look Through), при этом теряется по крайней мере один такт процессора, зато экономится энергия.

В современных компьютерах кеш обычно строится по двухуровневой схеме. **Первичный кеш** (*L1 Cache*) или кеш первого уровня, встроен во все процессоры на-

чиная с 486. Объем этого кеша невелик (от 8 до 64 Кбайт, иногда больше). Для повышения производительности для данных и команд часто используется раздельный кеш (так называемая Гарвардская архитектура — противоположность Принстонской, использующей общую память для команд и данных).

Вторичный кеш (*L2 Cache*) или кеш второго уровня, для процессоров 486 и Pentium является внешним, т.е. устанавливается на системной плате, а в Pentium Pro и всех последующих процессорах располагается в одной упаковке с ядром и подключается специальной внутренней шиной к процессору, или вообще является частью кристалла процессора.

Разница в расположении кеша очень существенная — в случае, если он находится на материнской плате, он работает на частоте системной (внешней) шины процессора, и доступ к нему осуществляется опять же через эту шину. А в случае с подключением кеша отдельной шиной, частота его работы и производительность этой шины не будет зависеть от системной шины, соответственно и производительность будет намного выше.

Кеш-контроллер должен обеспечивать когерентность (*coherency*) — согласованность данных кеш-памяти обоих уровней с данными в основной памяти.

Контроллер кеша оперирует строками (*cache line*) фиксированной длины. Строка может хранить копию блока основной памяти, размер которого, естественно, совпадает с длиной строки. С каждой строкой кеша связана информация об адресе скопированного в нее блока основной памяти и об ее состоянии. Строка может быть дей-

ствительной (*valid*) — это означает, что в текущий момент времени она достоверно отражает соответствующий блок основной памяти, или недействительной. Информация о том какой именно блок занимает данную строку (то есть старшая часть адреса или номер страницы), и о ее состоянии называется тегом (*tag*) и хранится в связанной с данной строкой ячейке специальной памяти тегов (tag RAM).

В операциях обмена с основной памятью обычно строка участвует целиком (несекторированный кеш). Возможен и вариант секторированного (*sectored*) кеша, при котором одна строка содержит несколько смежных ячеек — секторов, размер которых соответствует минимальной порции обмена данных кеша с основной памятью. При этом в записи каталога, соответствующей каждой строке, должны раниться биты действительности для каждого сектора данной строки.

Запись блока, не имеющего копии в кеше, производится в основную память (для повышения быстродействия запись может производиться через буфер отложенной записи). Поведение кеш-контроллера при операции записи в память, когда копия затребованной области находится в некоторой строке кеша, определяется его алгоритмом, или политикой записи (*Write Policy*). Существуют две основные политики записи данных из кеша в основную память: сквозная запись WT (*Write Through*) и обратная запись WB (*Write Back*).

Политика WT предусматривает выполнение каждой операции записи (даже однобайтной), попадающей в кешированный блок, одновременно и в строку кеша, и в основную память. При этом процессору при каждой опера-

ции записи придется выполнять относительно длительную запись в основную память. Алгоритм достаточно прост в реализации и легко обеспечивает целостность данных за счет постоянного совпадения копий данных в кеше и основной памяти. Для него не нужно хранить признаки присутствия и модифицированности — вполне достаточно только информации тега (при этом считается, что любая строка всегда отражает какой-либо блок, а какой именно — указывает тег). Но эта простота оборачивается низкой эффективностью записи. Существуют варианты этого алгоритма с применением отложенной буферизированной записи, при которой данные в основную память переписываются через FIFO-буфер во время свободных тактов шины.

Политика WB позволяет уменьшить количество операций записи на шине основной памяти. Если блок памяти, в который должна производиться запись, отображен в кеше, то физическая запись сначала будет произведена в эту действительную строку кеша, и она будет отмечена как грязная (*dirty*), или модифицированная, то есть требующая выгрузки в основную память. Только после этой выгрузки (записи в основную память) строка станет чистой (*clean*), и ее можно будет использовать для кеширования других блоков без потери целостности данных. В основную память данные переписываются только целой строкой. Эта выгрузка контроллером может откладываться до наступления крайней необходимости (обращение к кешированной памяти другим абонентом, замещение в кеше новыми данными) или выполняться в свободное время после модификации всей строки.

Эксклюзивный и не эксклюзивный кеш

Концепции эксклюзивного и не эксклюзивного кеширования очень просты: в случае не эксклюзивного кеша, информация на всех уровнях кеширования может дублироваться. Таким образом, L2 может содержать в себе данные, которые уже находятся в L1I и L1D, а L3 может содержать в себе полную копию всего содержимого L2 (и, соответственно, L1I и L1D). Эксклюзивный кеш, в отличие от не эксклюзивного, предусматривает чёткое разграничение: если информация содержится на каком-то уровне кеша — то на всех остальных она отсутствует. Плюс эксклюзивного кеша очевиден: общий размер кешируемой информации в данном случае равен суммарному объёму кешей всех уровней — в отличие от не эксклюзивного кеша, где размер кешируемой информации (в худшем случае) равен объёму самого большого уровня кеша. Минус эксклюзивного кеша менее очевиден, но он есть: необходим специальный механизм, который следит за собственно «эксклюзивностью» (так, например, при удалении информации из L1-кеша, перед этим автоматически инициируется процесс её копирования в L2).

Не эксклюзивный кеш традиционно использует компания Intel, эксклюзивный (с момента появления процессоров Athlon на ядре Thunderbird) — компания AMD. В целом, мы наблюдаем здесь классическое противостояние между объёмом и скоростью: за счёт эксклюзивности, при одинаковых объёмах L1/L2 у AMD общий размер кешируемой информации получается больше — но за счёт неё же он работает медленней (задержки, вызванные наличием механизма обеспечения эксклюзивности).

Другие наборы команд

Набор команд x86, который мы с Вами уже обсуждали, имеет следующую особенность — он ориентирован на работу с целыми числами. Как быть, если процессору нужно извлечь квадратный корень, найти синус или логарифм? Естественно, что процессор может справиться с такой задачей, но, учитывая то, что он ориентирован на вычисления с целыми числами, выполнение такой операции займет у него много тактов.

В то же время Intel для всех своих процессоров разрабатывает так называемый сопроцессор — кристалл, который тоже умеет выполнять команды, но не x86, а другие, и поддерживаемый сопроцессором набор команд (называемый x87) ориентирован на работу с числами с плавающей запятой, таким образом, он (сопроцессор) перечисленные выше задачи как раз и призван решать. На заре развития PC считалось, что сопроцессор нужен небольшому количеству пользователей (действительно, зачем пользователю текстового редактора математические вычисления) и устанавливался в систему дополнительно; при желании пользователь мог отдельно приобрести сопроцессор и установить его в специальное гнездо на материнской плате. Сегодня ситуация полностью изменилась.

Сегодня сопроцессор нужен абсолютно всем пользователям: «в наш век мультимедиа» когда компьютер все больше и больше способен обрабатывать реалистичную 3-х мерную графику, математические расчеты становятся неотъемлемым атрибутом любого мультимедиа приложения (например, современной игры). Поэтому достаточно

давно сопроцессор стали устанавливать вместе с процессором на один кристалл. Действительно, раз сопроцессор все равно нужен всем пользователям, то его разумно встроить в процессор, а не изготавливать отдельно: разумеется, это будет дешевле.

Таким образом, любой современный процессор поддерживает два основных набора команд: x86 и x87. Могут ли производители процессоров отказаться от этих наборов команд? Нет! Тогда получившаяся система уже не будет программно совместима с PC, так как программное обеспечение, написанное для PC на таком процессоре, уже не будет работать! Поддержка этих двух наборов команд — залог программной совместимости. А могут ли производители процессоров создавать и добавлять в процессор новые наборы команд? Конечно. Но что нужно для того, чтобы программы стали работать эффективнее на процессоре с новым набором команд? Будет ли у старого приложения, которое проектировали еще до разработки нового набора команд какие-то преимущества от исполнения на процессоре с новым набором команд? Нет! Ведь программа — это не что иное, как последовательность команд процессору. Если в программе нет ни одной новой команды (а откуда ей там взяться, если в момент написания программы новых команд еще не придумали), то естественно никакой пользы из того, что процессор может исполнять новые команды, старое приложение не извлечет. Поэтому, когда в процессор добавляют новый набор команд, нужно понимать, что пока разработчики программного обеспечения не начнут писать программы с учетом нового набора команд НИКАКОЙ

пользы от него не будет. А старым программам от нового набора команд уже никогда не будет пользы (разве что авторы перепишут старую программу с учетом нового набора команд). Процессор, изначально, задумывался как универсальное устройство, которое могло с помощью изменения программы, решать любые задачи, будь то решение математических уравнений, игра, программа набора текста и др. Но со временем, к некоторым областям стали предъявляться особые требования по быстродействию. Для решения, некоторых узких задач, были введены дополнительные наборы команд — расширения основного набора x86.

Технология MMX

В зависимости от контекста MMX может означать multi-media extensions (мультимедийные расширения) или matrix math extensions (матричные математические расширения). Технология MMX начала использоваться очень давно, благодаря этому набору команд ускоряется компрессия/декомпрессия видеоданных, манипулирование изображением, шифрование и выполнение операций ввода — вывода — почти все операции, используемые во многих современных программах. MMX это расширение основного набора команд на 57 новых команд, а также инновация состоит во введении новой возможности выполнения команд, называемой одиночный поток команд — множественный поток данных (*Single Instruction — Multiple Data*, SIMD), как ясно из названия одной командой из набора MMX можно обрабатывать много данных.

Набор инструкций 3DNow!/Enhanced 3DNow!

Технология 3DNow! разработана фирмой AMD для процессоров K6-2, конкурента Pentium MMX и Pentium II. Направлена эта технология, как видно из названия, на ускорение работы с трехмерной графикой, а также другие вычисления связанные с потоковыми видео- и аудиоданными, хотя по сути является просто математическим набором команд. Основная цель, которую преследовала AMD при создании набора команд 3DNow! — создание альтернативы x87, т.к. сопроцессоры, которые применялись фирмой AMD для своих процессоров, были недостаточно производительными в сравнении с сопроцессорами Intel, разработчиком x87, а трёхмерная графика и игры являются основными потребителями математических команд. И если заметить, именно игры для большинства пользователей являются двигателем к выбору аппаратной части компьютера, поэтому AMD сделали рискованный маркетинговый ход — внедрили новый набор команд в свои процессоры, и начали активно преподносить свои процессоры как наиболее оптимизированные под игровой процесс. Как мы уже говорили, успех того или иного набора команд целиком зависит от его поддержки программным обеспечением, поэтому для успеха своих процессоров AMD сделали ставку именно на игры. Ее дальнейшее развитие — Enhanced 3DNow!, появилась в процессорах Athlon и Duron.

Набор инструкций SSE/SSE2/SSE3

SSE (*Streaming SIMD Extensions* — потоковые расширения SIMD), обновление технологии MMX, появились в Pentium III. Содержит 70 новых инструкций для работы

с графикой и звуком в дополнение к существующим командам MMX. Инструкции SSE подобны инструкциям MMX и предварительно назывались MMX-2. Операции с плавающей точкой SSE реализованы в виде отдельного модуля в процессоре. Новые инструкции SSE позволяют более эффективно работать с трехмерной графикой, потоками аудио- и видеоданных, приложениями распознавания речи. Набор команд SSE2, который появились в Pentium 4 — это опять расширенный на новые 144 команды набор команд SSE, в первую очередь направлен на потоковые вычисления.

SSE3 был представлен в феврале 2004 года вместе с процессором Prescott Pentium 4 и добавляет 13 новых SIMD-инструкций для улучшения сложной математики, графики, кодирования видео и потока синхронизации. SSE3 также включает все предыдущие инструкции MMX, SSE и SSE2. SSSE3 (дополнительный SSE3) был представлен в июне 2006 года на серверных процессорах серии Xeon 5100 и в июле 2006 года в процессорах Core 2. SSSE3 добавляет 32 новых SIMD-инструкции к SSE3.

SSE4 (также называемый HD Boost от Intel) был представлен в январе 2008 года в версиях процессоров Intel Core 2 (SSE4.1) и позднее был обновлен в ноябре 2008 года на процессорах Core i7 (SSE4.2). SSE4 состоит из 54 полных инструкций с подмножеством из 47 инструкций, содержащих SSE4.1, и полные 54 инструкции в SSE4.2.

Блок схема узлов процессора

Теперь давайте попробуем объединить все выше перечисленные модули блоки и узлы в одну единую схему, и посмотрим что у нас с вами может получится. Если

за основу взять двухядерный процессор, мы с вами получим приблизительно следующее:

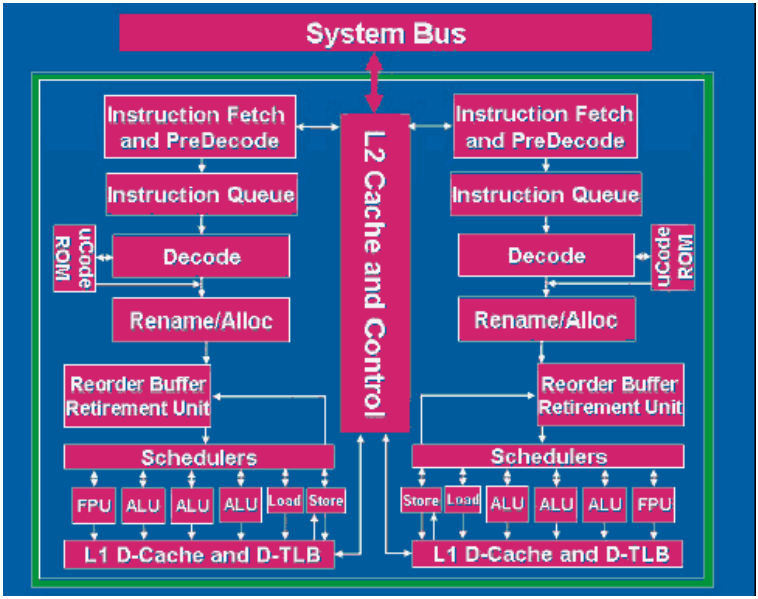

Рисунок 3

Многопроцессорные системы

Довольно давно возникла идея использования нескольких процессоров, вместо одного. Т.е. руководствуясь принципом «одна голова хорошо, а две лучше», производители процессоров решили найти способ увеличения производительности компьютера, не изменяя при этом параметров самих процессоров. Таким образом, стали появляться системы с 2-я, 4-я и более процессорами. По идее производительность таких систем должна быть больше в 2, 4 и т.д. раз, но на практике это совсем не так.

При создании многопроцессорных систем надо учитывать множество условий:

- Материнская плата должна уметь работать с несколькими процессорами, т.е. на ней должен быть соответствующий чипсет, и необходимое число разъёмов для процессоров.
- Сам процессор должен поддерживать работу в многопроцессорных системах,

Операционные системы и программное обеспечение также должны уметь работать с несколькими процессорами. В следующем курсе, мы будем ещё упоминать об этом, когда будем рассматривать возможности операционных систем. Многопроцессорные системы дают хороший прирост производительности, только тогда, когда на них выполняется специализированное программное обеспечение.

Компьютер с несколькими процессорами работает под управлением операционной системы, которая может распределять разные задачи по разным процессорам компьютера. А программы, которые выполняются на таком компьютере, должны состоять из нескольких потоков, которые можно выполнять независимо друг от друга. За счет этих возможностей будет увеличиваться производительность. Соответственно, если операционная система и программное обеспечение не отвечает этим требованиям, то и выигрыша от применения многопроцессорной системы не будет никакого, и её производительность будет такой же, что и у однопроцессорной системы. Существуют два режима работы многопро-

цессорных систем — симметричный и асимметричный. Рассмотрим их подробнее:

- **AMP, Asymmetric Multiprocessing.** Асимметричная многопроцессорность. В этом режиме операционная система выполняется на одних процессорах, а приложения выполняются другими процессорами. Такой режим в определённых условиях является неэффективным, т.к. нагрузка распределяется неравномерно, поэтому сейчас чаще всего применяется второй режим.
- **SMP, Symmetric Multiprocessing.** Симметричная многопроцессорность. В этом режиме и операционная система, и приложения выполняются любым процессором, в зависимости от его загрузки, что обеспечивает большую гибкость при распределении нагрузки, и соответственно большую производительность.

У многопроцессорных систем есть кроме множества достоинств, и явный недостаток — стоимость. Материнские платы для многопроцессорных систем стоят намного дороже однопроцессорных, да и покупка нескольких процессоров тоже не является дешевым удовольствием. Поэтому производители стали разрабатывать более дешевые варианты многопроцессорности.

Первый из этих вариантов, который появился в процессорах Pentium 4, это технология *Hyper Threading* (HT), которая представляет из себя «логическую многопроцессорность». В чём её особенность?

Как вы уже знаете, процессор состоит из нескольких блоков, выполняющих различные задачи, например 2 конвейера ALU, из которых один основной а второй вспомо-

гательный, т.е. загружен не всегда, 1 FPU и блок загружающий и выгружающий команды и данные из памяти. Представьте ситуацию, что во время выполнения программы возникает ситуация, когда на протяжении работы нескольких стадий конвейера загружены 60% только ALU, FPU не загружен вообще и блок загрузки-выгрузки занят на 30%. В среднем выйдет процент использования ресурсов процессора примерно 30%. Но это только конкретный случай, возможны и масса других ситуаций, когда процессор будет использовать большее, или меньшее число своих блоков, и во многом это зависит от программного обеспечения. Сама Intel заявляет, что блоки процессора Pentium 4 загружены в среднем не более чем на 35%. Поэтому они и предложили технологию, которая при небольших изменениях ядра (увеличение примерно на 10%), позволяет использовать простаивающие блоки процессора в качестве дополнительного, 2-го процессора. Для этого был добавлен ещё один блок регистров, и программы работают с таким процессором не как с одним, и как с двумя. Конечно, такой подход не даст полноценной альтернативы многопроцессорным системам, но в зависимости от конкретного применения HT, прирост производительности мог достигать 30%.

Также, с уменьшением технологического процесса производства процессоров, появилась возможность создавать на одном кристалле, который помещается в стандартный корпус 2 и более процессоров. Такие процессоры называются многоядерными. А такую архитектуру называют CMP, Cellular MultiProcessing, сотовая мультипроцессорность. Особенностью, и отличием такой архи-

тектуры от SMP, является использование несколькими ядрами общего пространства памяти.

Производители процессоров в настоящее время выпускают 2-х и 4-х и 6-и ядерные процессоры для рабочих станций, и процессоры с большим количеством ядер для серверов. На фото изображение двухъядерного процессора, и можно чётко различить два одинаковых ядра.

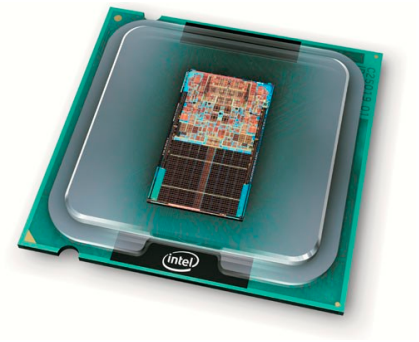

Рисунок 4

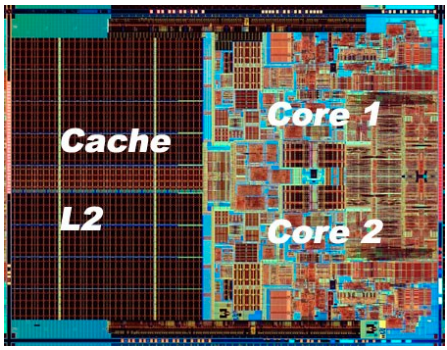

Рисунок 5

Режимы работы процессора

Все 32-разрядные и более поздние процессоры Intel, начиная с 386, могут выполнять программы в нескольких режимах. Режимы процессора предназначены для выполнения программ в различных средах; в разных режимах работы возможности чипа не одинаковы, потому, что команды выполняются по-разному. В зависимости от режима процессора изменяется схема управления памятью системы и задачами.

Процессоры могут работать в трех режимах: реальном, защищенном и виртуальном реальном режиме (реальном внутри защищенного).

Реальный режим

В первоначальном IBM PC использовался процессор 8088, который мог выполнять 16-разрядные команды, используя 16-разрядные внутренние регистры и адресовать только 1 Мб памяти, используя 20-и разрядную шину адреса. Все программное обеспечение PC первоначально было предназначено для этого процессора, оно было разработано на основе 16-разрядной системы команд и модели памяти, объемом 1 Мб. Например DOS, все программное обеспечение DOS написано в расчете на 16-разрядные команды. Более поздние процессоры, например 286, могли также выполнять те же самые 16-разрядные команды, что и первоначальный 8088, но намного быстрее.

Другими словами процессор 286 был полностью совместим с первоначальным 8088. 16-разрядный режим, в котором выполнялись команды процессоров 8088 и 80286 был назван реальным режимом. Все программы, выполняющиеся в реальном режиме, должны использовать только 16-разрядные команды и 20-разрядный адрес. Для программного обеспечения такого типа используется однозадачный режим, т.е. одновременно должна выполняться только одна программа. Нет никакой встроенной защиты для предотвращения перезаписи ячеек памяти, занятых одной программой или даже самой операционной системой, другими программами:

Это означает, что при выполнении нескольких программ вполне могут быть испорчены данные или код одной из программ, что может привести к остановке системы.

Защищенный режим

Первым 32-разрядным процессором, предназначенным для PC, был 386-ой. Этот чип мог выполнять абсолютно новую 32-разрядную систему команд. Для того чтобы полностью использовать преимущество этой новой системы команд, были необходимы 32-разрядная операционная система и 32-разрядные приложения. Этот новый режим называли защищенным, так как выполняющиеся в нем программы защищены от перезаписи используемых ими областей памяти другими программами.

Такая защита делает систему более надежной, так как уже ни одна программа с ошибками не сможет повредить другие программы или операционную систему. Зная, что разработка новых операционных систем и приложений, использующих преимущество 32-разрядного защищенного режима, займет некоторое время, Intel предусмотрела в процессоре 386 обратно совместимый реальный режим. Благодаря этому процессор 386 мог выполнять обычные 16-разрядные приложения и операционные системы. Причем они выполнялись намного быстрее, чем на любом процессоре предыдущего поколения. Для большинства пользователей этого было достаточно: 32-разрядные системы и приложения остались на тот момент не востребованы и потребители довольствовались тем, что уже имеющиеся у них 16-разрядные программы работали быстрее. К сожалению, из-за этого 386 процессор так никогда и не использовался в защищенном режиме, и, стало быть, все преимущества такого режима терялись. Когда современный высокопроизводительный процессор работает в реальном режиме, то он напоми-

нает чудовищно ускоренный 8088! Т.е. процессор может хоть и с огромной (по сравнению с оригинальным 8088) скоростью, но все же выполнять только 16-разрядные приложения и адресоваться только к памяти размером 1 Мб, с которой мог работать 8088. Поэтому были необходимы новые операционные системы и новые приложения, которые могли бы на современных процессорах выполняться в защищенном 32-разрядном режиме. Однако пользователи сопротивлялись всем попыткам перехода к 32-разрядной среде. Для них это означало, что нужно, по крайней мере частично, отказываться от старого программного обеспечения.

Только в августе 1995 года (спустя 10 лет после выхода первого 32-разрядного процессора) наконец появилась первая пользовательская 32-разрядная операционная система Windows 95, да и то, пользователи приняли ее во многом потому, что она частично 16-разрядная и поэтому без труда исполняет как новые 32-разрядные программы, так и старые, 16-разрядные. Именно для такой обратной совместимости Windows 95 использовала третий режим процессора:

Виртуальный реальный режим

Виртуальный реальный, по существу, является режимом выполнения 16-разрядной среды (реальный режим), который реализован внутри 32-разрядного защищенного режима. Поскольку защищенный режим является подлинно многозадачным, фактически можно выполнять несколько сеансов реального режима, причем в каждом сеансе собственное программное обеспечение выполня-

ется на собственном виртуальном компьютере. И все эти приложения могут выполняться одновременно, даже во время выполнения других 32-разрядных программ.

Следует обратить внимание на то, что любая программа, выполняющаяся в виртуальном реальном режиме, может обращаться к памяти, объемом до 1 Мб, причем для каждой такой программы это будет как бы первый и единственный мегабайт памяти в системе. Виртуальное реальное окно полностью имитирует среду процессора 8088и если не учитывать быстродействие, программное обеспечение в виртуальном реальном режиме выполняется так, как выполнялось бы на самых первых PC в реальном режиме. При запуске каждого 16-разрядного приложения Windows 95/98 создает так называемую виртуальную машину DOS, выдает ей 1 Мб памяти и на этой машине 16-разрядное приложение выполняется. Следует обратить внимание на то, что все процессоры при включении начинают работать в реальном режиме, и только при старте 32-разрядной операционной системы происходит переключение в 32-разрядный режим.

64 разрядный расширенный режим IA-32e (AMD64, x86-64, EM64T)

Этот режим является расширением архитектуры IA-32, разработанным компанией AMD и в дальнейшем поддержанным Intel. Процессоры, поддерживающие 64-разрядные расширения, могут работать в реальном режиме (8086), режиме IA-32 или IA-32e. При использовании режима IA-32 процессор может работать в защищенном или виртуальном реальном режиме.

Режим IA-32e позволяет работать в 64-х разрядном режиме или в режиме совместимости, что подразумевает возможность одновременного выполнения 64 и 32-х разрядных приложений.

- **64-х разрядный режим.** Позволяет 64-х разрядной операционной системе выполнять 64-х разрядные приложения.
- **Режим совместимости.** Позволяет 64-х разрядной операционной системе выполнять 32-х разрядные приложения.

Режим совместимости IE-32e позволяет запускать 32-х и 16-и разрядные приложения под управлением 64-х разрядной операционной системы.

Системная плата

Одним из важнейшим узлом компьютера является системная плата (*System Board*), иногда называемая материнской (*Motherboard*), основной или главной платой (*Main Board*). В этом уроке мы с вами рассмотрим типы системных плат и их компоненты.

Так зачем же она нам нужна? С одной стороны мы уже с вами знаем, что системный блок нашего с вами компьютера, состоит из большего количества устройств. Это процессор, оперативная память, постоянные запоминающие устройства, а также огромное количество другого дополнительного оборудования, которое нужно правильно объединить. Обеспечением передачи данных между ними и занимается материнская плата. Отсюда определение материнской платы:

Материнская плата (*Motherboard*) — основной элемент системного блока, позволяющий объединить все основные компоненты и устройства компьютера, и обеспечить передачу данных между ними.

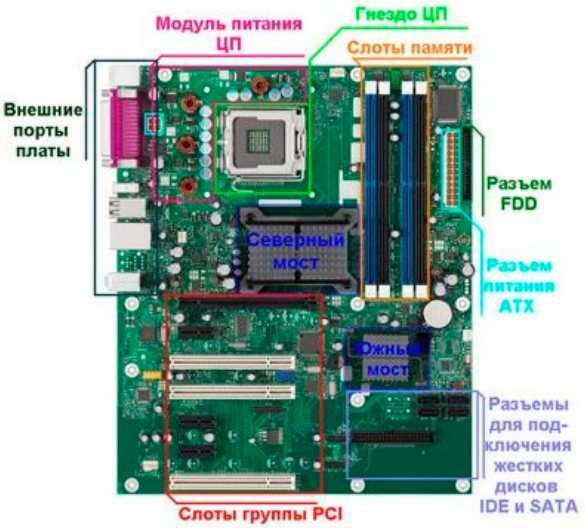

Рисунок 6. Пример схемы материнской платы

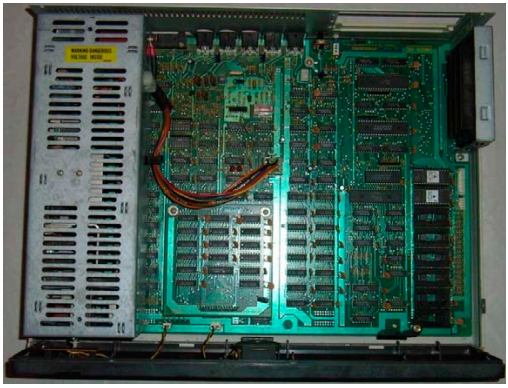

Рисунок 7. Материнская плата стандарта XT/AT (IBM PC 1981 год вып.)

Еще 25 лет назад системные платы (и подавляющее большинство других узлов) персональных компьютеров строились на основе цифровых микросхем малой и средней степени интеграции (вентилей, триггеров, регистров и т.п.). И если бы вам пришлось иметь дело с компьютерами XT/AT, тогда вы бы увидели системную плату с полутора-двумя сотнями корпусов интегральных схем (ИС).

Эти микросхемы потребляли значительную мощность и занимали много места на платах. Разработчики достаточно быстро поняли, что стандартные узлы персональных компьютеров — контроллеры накопителей на гибких дисках, прямого доступа к памяти (DMA— *Direct Memory Access*), программируемые контроллеры прерываний — можно выполнить в виде специализированных интегральных схем (СИС или ASIC — *Application Specific Integrated Circuit*).

Использование таких СИС позволило резко сократить количество элементов и межвыводных соединений на платах компьютеров, снизить их стоимость и потребляемую мощность. Высокая степень интеграции микросхем привела к повышению производительности компьютеров. За счет размещения сложных логических схем всего лишь в нескольких корпусах удалось значительно сократить длину сигнальных проводников (а, значит, и емкости монтажа), что позволило повысить рабочие частоты логических схем. Оптимизация разводки проводников в самих микросхемах привела к еще большему увеличению скорости выполнения операций. Разработчики довольно быстро сообразили, что, в конечном счете, все необходимые для работы современных компьютеров системные контроллеры можно «упаковать» в несколько микросхем, с высокой

степенью интеграции. Поскольку такие микросхемы разрабатывались как наборы, предназначенные для построения системных плат, их стали называть комплектами ИС (интегральные схемы) или Чипсетами (*Chipset*).

В настоящее время чипсеты играют определяющую роль в проектировании и производстве персональных компьютеров. Если на первых системных платах, как уже отмечалось ранее, устанавливались сотни микросхем, то сейчас их количество очень редко превышает два десятка. Роль чипсетов системных плат настолько велика, что, как правило, с появлением какой-либо новой технологии или процессора приходится разрабатывать их новые разновидности.

В конечном счете, суммарные возможности персонального компьютера в значительной степени определяются чипсетом системной платы. Чипсет образует костяк любой компьютерной системы.

Один из первых чипсетов материнской платы состоял из двух компонентов (которые представляли собой независимые чипсеты, связанные друг с другом). Назывались эти компоненты Северный и Южный мост. Названия Северный и Южный —означали расположение чипсета моста относительно шины PCI: Северный находится выше, а Южный — ниже. Почему мост? Это название дали чипсетам по выполняемыми ими функциями: они служат для связи различных шин и интерфейсов. Северный Мост работал с самыми скоростными устройствами, обеспечивая быструю и надежную связь процессора, памяти, шины AGP и Южного Моста.

Южный мост работал с медленными устройствами, такими как жесткие диски, шина USB, PCI, и т.п.

Intel использовала три различные архитектуры чипсетов:

- **Северный / южный мост** (использовался чипсетами 400-й серии).
- **Хаб** (использовался чипсетами 800-й серии, 900-й серии, 3x, 4x и 5x).
- **Одночиповая** (используется чипсетами серий 6x, 7x, 8x и 9x и другими современными чипсетами).

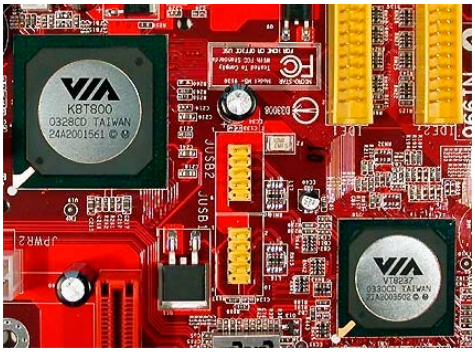

Рисунок 8. Северный и южный мосты материнской платы

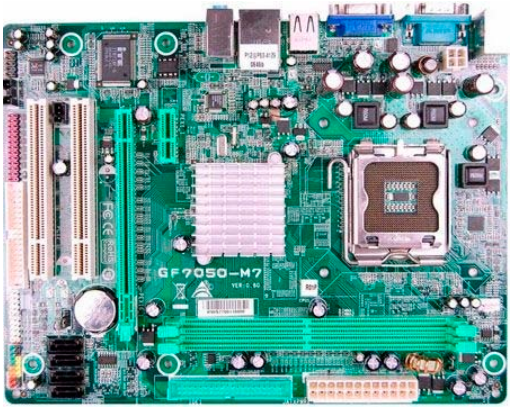

Рисунок 9. Материнская плата с чипсетом от nVidia

Урок №2
Центральный процессор