Лекция 3

Функциональная схема ЭВМ

1. Архитектура классической ЭВМ
2. Принципы построения
3. Адресация памяти
4. Машинные команды
5. Порядок выполнения машинной программы
6. RISC – процессор
7. Пятиступенчатый конвейер для обработки команд
8. Многоядерный процессор

Структура классической ЭВМ

В 1946 году Джоном Нейманом на летней сессии Пенсильванского университета был распространен отчет, заложивший основы развития вычислительной техники на несколько десятилетий вперед. Последующий опыт разработки ЭВМ показал правильность основных выводов Неймана, которые, естественно, в последующие годы развивались и уточнялись.

Основные рекомендации, предложенные Нейманом для разработчиков ЭВМ

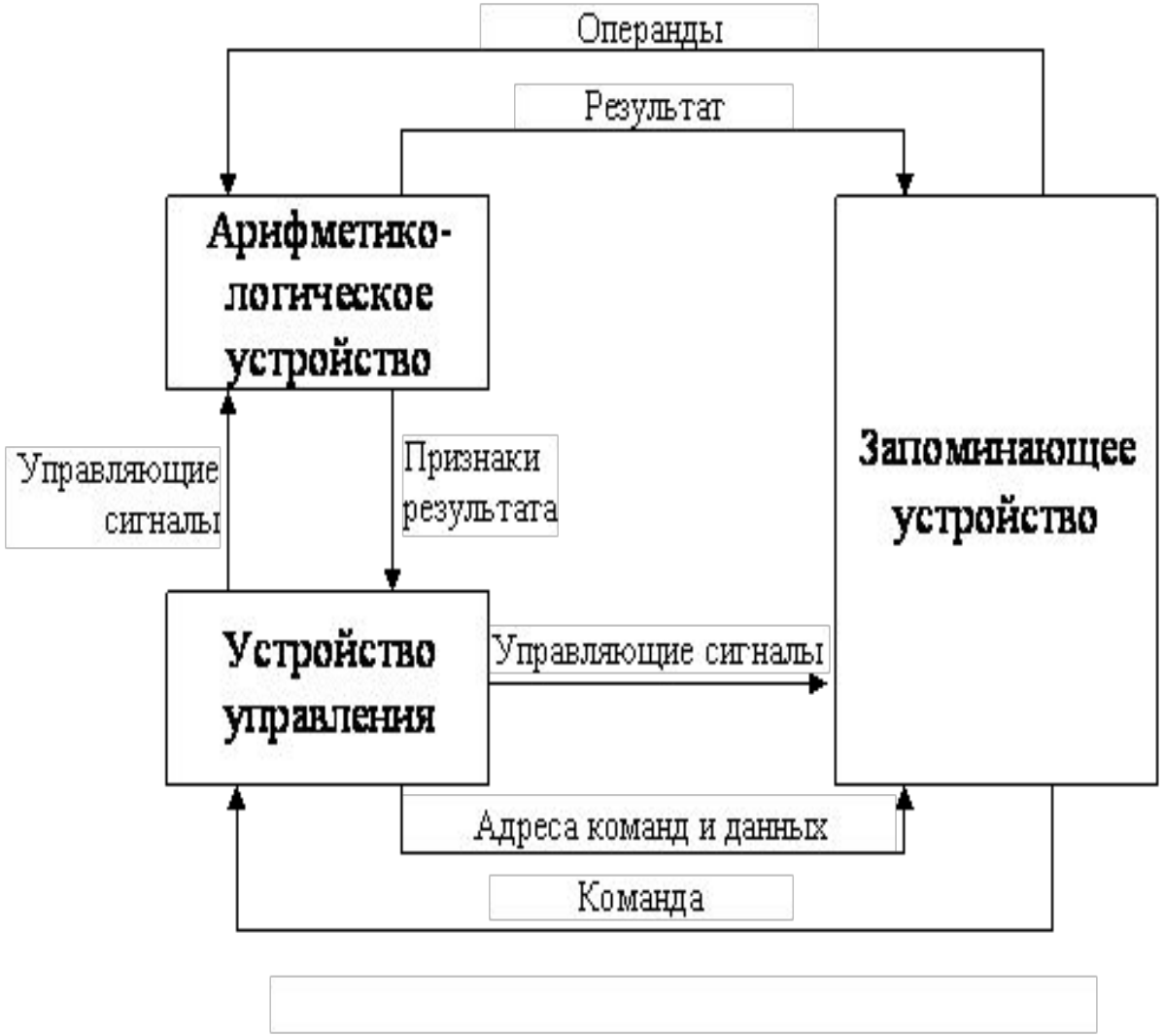
1. Машины на электронных элементах должны работать не в десятичной, а в двоичной системе счисления.
2. Программа должна размещаться в одном из блоков машины – в *запоминающем устройстве* (ЗУ), обладающем достаточной емкостью и соответствующими скоростями выборки и записи команд программы.
3. Программа так же, как и числа, с которыми оперирует машина, представляется в двоичном коде. Таким образом, по форме представления команды и числа однотипны. Это обстоятельство приводит к следующим важным последствиям:
    a. промежуточные результаты вычислений, константы и другие числа могут размещаться в том же ЗУ, что и программа;
    b. числовая форма записи программы позволяет машине производить операции над величинами, которыми закодированы команды программы.

Основные рекомендации, предложенные Нейманом для разработчиков ЭВМ (продолжение)

4. Трудности физической реализации ЗУ, быстродействие которого соответствовало бы скорости работы логических схем, требует иерархической организации памяти.
5. Арифметические устройства машины конструируются на основе схем, выполняющих операцию сложения. Создание специальных устройств для вычисления других операций нецелесообразно.
6. В машине используется параллельный принцип организации вычислительного процесса (операции над словами производятся одновременно по всем разрядам).

Структура классической ЭВМ.

ЭВМ, построенная по принципам, определенным Нейманом, состоит из следующих основных блоков: *запоминающего устройства*, *арифметико-логического устройства* и *устройства управления*



Запоминающее устройство. Машинная команда

Запоминающее устройство, или память – это совокупность ячеек, предназначенных для хранения некоторого кода. Каждой из ячеек присвоен свой номер, называемый **адресом** . Информацией, записанной в ячейке, могут быть как команды в машинном виде, так и данные.

Машинная команда – это двоичный код, определяющий выполняемую операцию, адреса используемых операндов и адрес ячейки ЗУ, по которому должен быть записан результат выполненной операции.

Операции, определяемые кодом операции команды, выполняются в *арифметико-логическом устройстве* (АЛУ).

Устройство управления

Все действия в ЭВМ выполняются под управлением сигналов, вырабатываемых *устройством управления* (УУ). Управляющие сигналы формируются на основе информации, содержащейся в выполняемой команде, и признаков результата, сформированных предыдущей командой (если выполняемая команда является, например, командой условного перехода). *Устройство управления* помимо сигналов, определяющих те или иные действия в различных блоках ЭВМ (например, вид операции в АЛУ или сигнал считывания из ЗУ), формирует также адреса ячеек, по которым производится обращение к памяти для считывания команды и операндов и записи результата выполнения команды.

Устройство управления

- *Устройство управления* формирует адрес команды, которая должна быть выполнена в данном цикле, и выдает управляющий сигнал на чтение содержимого соответствующей ячейки *запоминающего устройства*. Считанная команда передается в УУ. По информации, содержащейся в адресных полях команды, УУ формирует адреса операндов и управляющие сигналы для их чтения из ЗУ и передачи в *арифметико-логическое устройство*. После считывания операндов *устройство управления* по коду операции, содержащемуся в команде, выдает в АЛУ сигналы на выполнение операции. Полученный результат записывается в ЗУ по адресу приемника результата под управлением сигналов записи.

Организация оперативной памяти (на примере 16-битовой ЭВМ)

Оперативная память является основной памятью для хранения информации. Она организована как одномерный массив ячеек памяти размером в 1 байт.

Каждый из байтов имеет уникальный 20 битный *физический адрес* в диапазоне от 00000 до FFFFFh (здесь и далее для записи адресов используется шестнадцатеричная система счисления, признаком которой является символ h в конце кода).

Таким образом, размер адресного пространства оперативной памяти составляет 2^{20} = 1Мбайт.

Принцип ЗМ

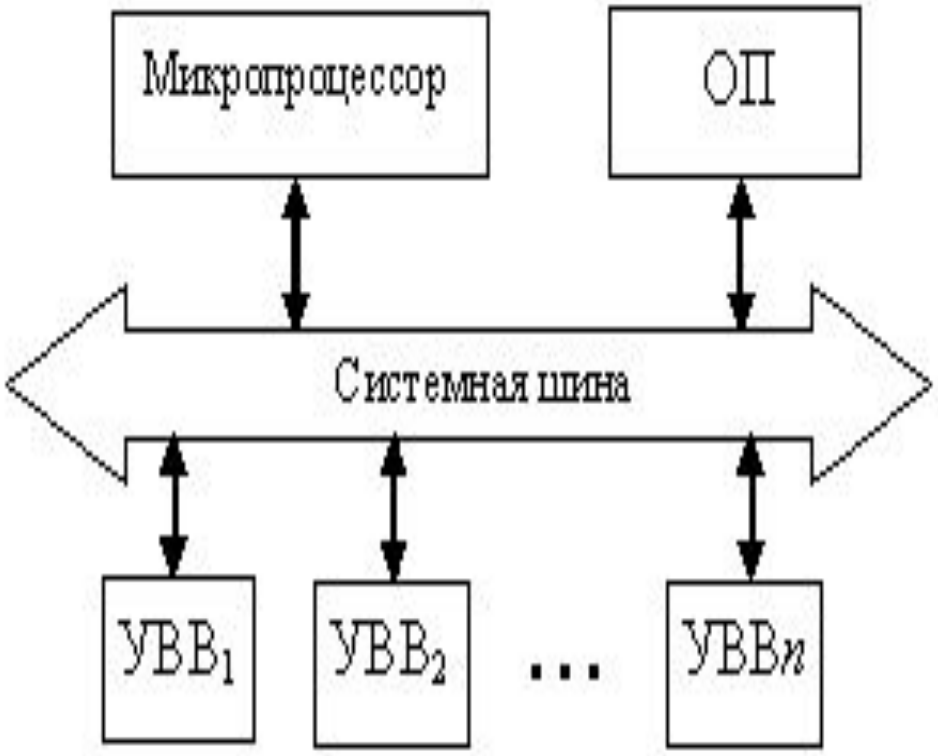
Компьютер строится на базе МОДУЛЬНОЙ структуры, т.е. Все основные компоненты представляют собой отдельные взаимозаменяемые модули.

МИКРОПРОГРАММИРУЕМОСТЬ – аппаратно-программная реализация основных типовых операций работы ЭВМ.

Компьютер строится по МАГИСТРАЛЬНО-модульному принципу, при котором все блоки компьютера связываются между собой системной шиной, предназначенной для обмена данными, адресной и управляющей информацией между составными частями ЭВМ. Как правило, при такой организации в любой момент может быть установлена связь только между двумя модулями ЭВМ.

Структура персональной ЭВМ

- *Персональная ЭВМ* типа IBM PC включает в себя *микропроцессор* (МП), оперативную память и устройства ввода-вывода (УВВ), объединенные между собой системной шиной.



Системная шина

Системная шина определяет общий порядок обмена между любыми блоками компьютера, а также максимальное количество используемых устройств ввода-вывода. Она включала в себя шину адреса (ША), шину данных (ШД) и шину управления (ШУ), содержащую набор линий, по которым передаются управляющие сигналы между блоками компьютера.

Специфика каждого конкретного блока учитывается особыми управляющими устройствами – контроллерами, входящими в состав этих блоков, например, работой жесткого диска управляет контроллер жесткого диска, используя информацию, поступающую к нему от *микропроцессора* по системной магистрали.

Прерывания (APIC)

APIC (Advanced Programmable Interrupt Controller) – одна из важнейших «деталек» любого компьютера. Это небольшая схема, занимающаяся сбором и обработкой возникающих в компьютере прерываний. Скажем, нажал пользователь клавишу на клавиатуре – контроллер клавиатуры зафиксировал это событие, занес код нажатой клавиши в свою встроенную память – и сгенерировал прерывание – выдал по специальной линии сигнал-запрос с просьбой прервать выполнение текущей программы и обработать событие «нажата клавиша на клавиатуре». Это, так сказать, «классическая» задача Interrupt Controller-ов: они позволяют процессору не терять зря время, регулярно опрашивая каждое из устройств на предмет того, «а не случилось ли там чего-нибудь за последние 10 мс».

Вектор прерываний

Но задачи ІС не ограничиваются только этим: помимо аппаратных существуют еще и программные прерывания (exceptions), которые генерирует не периферия, а сам процессор – в случае возникновения какой-либо нештатной ситуации. Типичные примеры – в программе встретилась непонятно какая, или просто запрещенная «простому пользователю» инструкция (#GP, General Protection Exception), произошло деление на ноль (#DE, Divide-by-Zero Error Exception), программа обратилась к несуществующему адресу в памяти (#PF, Page Fault Exception). Реакция на каждое из прерываний задается так называемым вектором прерываний – набором адресов в памяти, описывающих «что делать дальше» процессору в случае возникновения прерывания: какие функции (обработчики прерывания) ему в этом случае необходимо выполнять.

Прямой доступ в память (DMA)

DMA (Direct Memory Access) – это такой своеобразный «альтернативный процессор», который занимается в чипсете обработкой «фоновых» задач, связанных с периферией. Скажем, если процессору требуется прочитать пару килобайт данных с жесткого диска, то ему вовсе не обязательно терпеливо ждать целую вечность (несколько миллисекунд), пока эти самые данные ему не будут предоставлены. Вместо этого он может запрограммировать DMA-контроллер, чтобы тот выполнил эту задачу за него, и переключиться, пока этот запрос выполняется, на какую-нибудь другую задачу. Штука это не столь незаменимая, как APIC, но без неё не было бы даже интерфейса Ultra ATA/33

Таблица адресов графики (GART)

GART (Graphical Address Relocation Table) появился в компьютерах одновременно с шиной AGP: это небольшая схема, которая обеспечивает графическому ускорителю доступ к системной памяти процессора. Её задачи – реализация механизма виртуальной памяти для GPU, то есть отображение «линейного» адресного пространства, с которым работает ускоритель, на «реальное», произвольным образом «перетасованное» с «обычными данными». Позволяет современным 3D-ускорителям использовать не только «набортную» видеопамять, но и «основную» системную память компьютера.

Компоненты персонального компьютера

- **Системный блок** Блок питанияБлок питания • ОхлаждениеБлок питания • Охлаждение • Материнская платаБлок питания • Охлаждение • Материнская плата • ПроцессорБлок питания • Охлаждение • Материнская плата • Процессор • ШиныБлок питания • Охлаждение • Материнская плата • Процессор • Шины • ВидеокартаБлок питания • Охлаждение • Материнская плата • Процессор • Шины • Видеокарта • Звуковая платаБлок питания • Охлаждение • Материнская плата • Процессор • Шины • Видеокарта • Звуковая плата • Сетевая плата
- **Память** Оперативная памятьОперативная память • Запоминающее устройство с произвольным доступом
- **Носители информации** Жёсткий дискЖёсткий диск • Твердотельный накопительЖёсткий диск • Твердотельный накопитель (Флеш-памятьЖёсткий диск • Твердотельный накопитель (Флеш-память • USB-флешЖёсткий диск • Твердотельный накопитель (Флеш-память • USB-флеш) • Оптический приводЖёсткий диск • Твердотельный накопитель (Флеш-память • USB-флеш) • Оптический привод (CDЖёсткий диск • Твердотельный накопитель (Флеш-память • USB-флеш) • Оптический привод (CD • DVDЖёсткий диск • Твердотельный накопитель (Флеш-память • USB-флеш) • Оптический привод

Система кодирования команд

Запись любой команды определяется ее форматом. **Формат команды** – это структура команды, позволяющая распознать назначение отдельных ее полей.

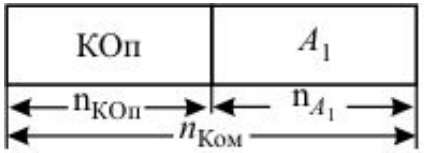
Исходя из определения, команда должна содержать информацию о выполняемой операции, адресах операндов и адресе ячейки ЗУ для записи результата. Этому в наибольшей степени соответствует формат команды, содержащий поле кода операции и три адресных поля. Такая система *кодирования команд* называется **трехадресной**

Схема выполнения трехадресной команды имеет вид:

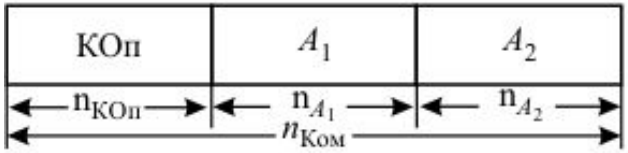
(A1)*(A2)->A3.

Здесь (A1) и (A2) – адреса ячеек ЗУ, в которых хранятся первый и второй операнды соответственно; * – знак обобщенной операции (например, сложение или умножение), задаваемой полем кода операции (КОп). Знак "->" обозначает передачу результата операции в ячейку памяти с адресом A3.

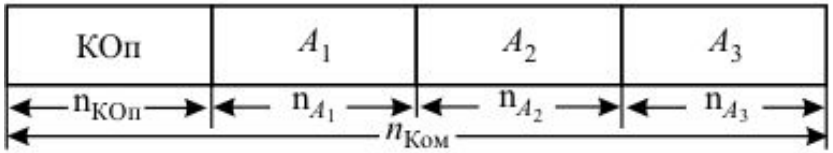
Виды команд



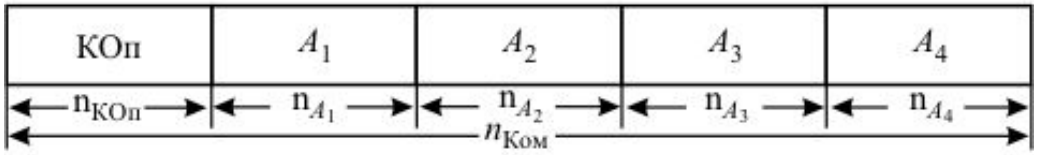
а) одноадресная команда



б) двухадресная команда



в) трехадресная команда



г) четырехадресная команда

Форматы команд

Рассмотренные форматы команд используются при так называемом **естественном порядке** выполнения программы. При этом подразумевается, что после выполнения любой команды, не меняющей в явном виде порядок выполнения программы, очередная команда выбирается из ячейки ЗУ, располагающейся сразу же вслед за ячейкой (или ячейками), содержащей код текущей команды. При **четырехадресной** системе *кодирования команд* (первые три адреса выполняют те же функции, что и в трехадресной команде, а четвертый адрес указывает адрес ячейки, где хранится следующая выполняемая команда. Такая система обеспечивает **принудительный порядок** выполнения команд программы. Она хотя и повышает гибкость программирования, но практического применения не получила. Основной причиной этого является существенное увеличение размера каждой команды и, соответственно, увеличение объема ЗУ, необходимого для размещения программы, в то время как реальной потребности в такой кодировке каждой команды не существует.

Регистровая память

Одним из способов уменьшения длины поля адреса является введение в состав ЭВМ дополнительно специального блока памяти небольшого объема – регистровой памяти (РП). Это запоминающее устройство имеет высокое быстродействие и служит для хранения часто используемой информации: промежуточных результатов вычислений, счетчиков циклов, составляющих адреса при некоторых режимах адресации и т.д..

Так как объем РП невелик, адресация ее элементов требует относительно короткого адресного поля. Например, для регистровой памяти объемом 8 регистров требуется всего лишь трехразрядное адресное поле.

Регистровая память

Решить проблему сокращения разрядности команды только за счет сокращения количества указываемых в команде операндов и применения регистровой памяти невозможно. Этой же цели служит использование различных *способов адресации* операндов. Кроме того, применение нескольких *способов адресации* повышает гибкость программирования, так как в каждом конкретном случае позволяет обеспечить наиболее рациональный способ доступа к информации в памяти.

Способы адресации

Различные *способы адресации* базируются на разных механизмах определения физического адреса операнда, то есть адреса фактического обращения к памяти при выполнении команды. Определение набора *способов адресации*, закладываемых в систему команд, является одним из важнейших вопросов разработки ЭВМ, существенно влияющим на ее архитектуру, вычислительные возможности, объем оборудования, быстродействие и другие характеристики.

К **основным** *способам адресации* относятся следующие: прямая, непосредственная, косвенная, относительная.

Прямая адресация

Физический адрес операнда совпадает с кодом в адресной части команды . Формальное обозначение:

$$\text{Операнд}_i = (A_i),$$

где A_i – код, содержащийся в i-м адресном поле команды.

При описании способов *кодирования команд* и расчете длины адресного поля предполагалось использование именно этого *способа адресации*.

Допускается использование *прямой адресации* при обращении как к основной, так и к регистровой памяти.

Непосредственная адресация

В команде содержится не адрес операнда, а непосредственно сам операнд:

$$Операнд_i = A_i.$$

Непосредственная адресация позволяет повысить скорость выполнения операции, так как в этом случае вся команда, включая операнд, считывается из памяти одновременно и на время выполнения команды хранится в процессоре в специальном регистре команд (РК). Однако при использовании непосредственной адресации появляется зависимость кодов команд от данных, что требует изменения программы при каждом изменении непосредственного операнда.

Косвенная адресация

Адресная часть команды указывает адрес ячейки памяти или номер регистра, в которых содержится адрес операнда:

$$Операнд_i = ((A_i)).$$

Применение *косвенной адресации* операнда из оперативной памяти при хранении его адреса в регистровой памяти существенно сокращает длину поля адреса, одновременно сохраняя возможность использовать для указания физического адреса полную разрядность регистра.

Недостаток этого способа – необходимо дополнительное время для чтения адреса операнда. Вместе с тем он существенно повышает гибкость программирования. Изменяя содержимое ячейки памяти или регистра, через которые осуществляется адресация, можно, не меняя команды в программе, обрабатывать операнды, хранящиеся по разным адресам.

Косвенная адресация не применяется по отношению к операндам, находящимся в регистровой памяти.

Относительная адресация

Этот способ используется тогда, когда память логически разбивается на блоки, называемые сегментами. В этом случае адрес ячейки памяти содержит две составляющих: адрес начала сегмента (базовый адрес) и смещение адреса операнда в сегменте. Адрес операнда определяется как сумма базового адреса и смещения относительно этой базы:

$$Операнд_i = (база_i + смещение_i).$$

Для задания базового адреса и смещения могут применяться ранее рассмотренные *способы адресации*. Как правило, базовый адрес находится в одном из регистров регистровой памяти, а смещение может быть задано в самой команде или регистре.

Главный недостаток относительной адресации

Главный недостаток *относительной адресации* – большое время вычисления физического адреса операнда. Но существенное преимущество этого *способа адресации* заключается в возможности создания "перемещаемых" программ – программ, которые можно размещать в различных частях памяти без изменения команд программы. То же относится к программам, обрабатывающим по единому алгоритму информацию, расположенную в различных областях ЗУ.

В этих случаях достаточно изменить содержимое базового адреса начала команд программы или массива данных, а не модифицировать сами команды. По этой причине *относительная адресация* облегчает распределение памяти при составлении сложных программ и широко используется при автоматическом распределении памяти в мультипрограммных вычислительных системах.

Порядок выполнения программы в кодах ЭВМ

(Адреса в шестнадцатеричной системе счисления)

№ команды п/п	КОП	A_1	A_2	A_3
1	01	0091	00A5	F003
2	03	F003	00C3	104D
. . .	. . .	. . .	. . .	. . .
n	02	0031	A32E	D006

Понятие параллелизма в вычислительном процессе

В 50-х годах исследования вычислительного процесса показали, что часть команд программы может выполняться одновременно и независимо друг от друга.

Другими словами – *параллельно.* И с тех пор история вычислительной техники развивалась в соответствии с логикой расширения параллелизма.

Что значит параллельно? – пример.

Что значит параллельно?

C = A + B

D = E + F

независимые данные, участки программ.

Сложение двух векторов A(i,n), B(i,n), цикл

```
      i=1
10    C(i)= A(i)+ B(i)
      i=i+1
      if  i  ≤ n Then go to 10
```

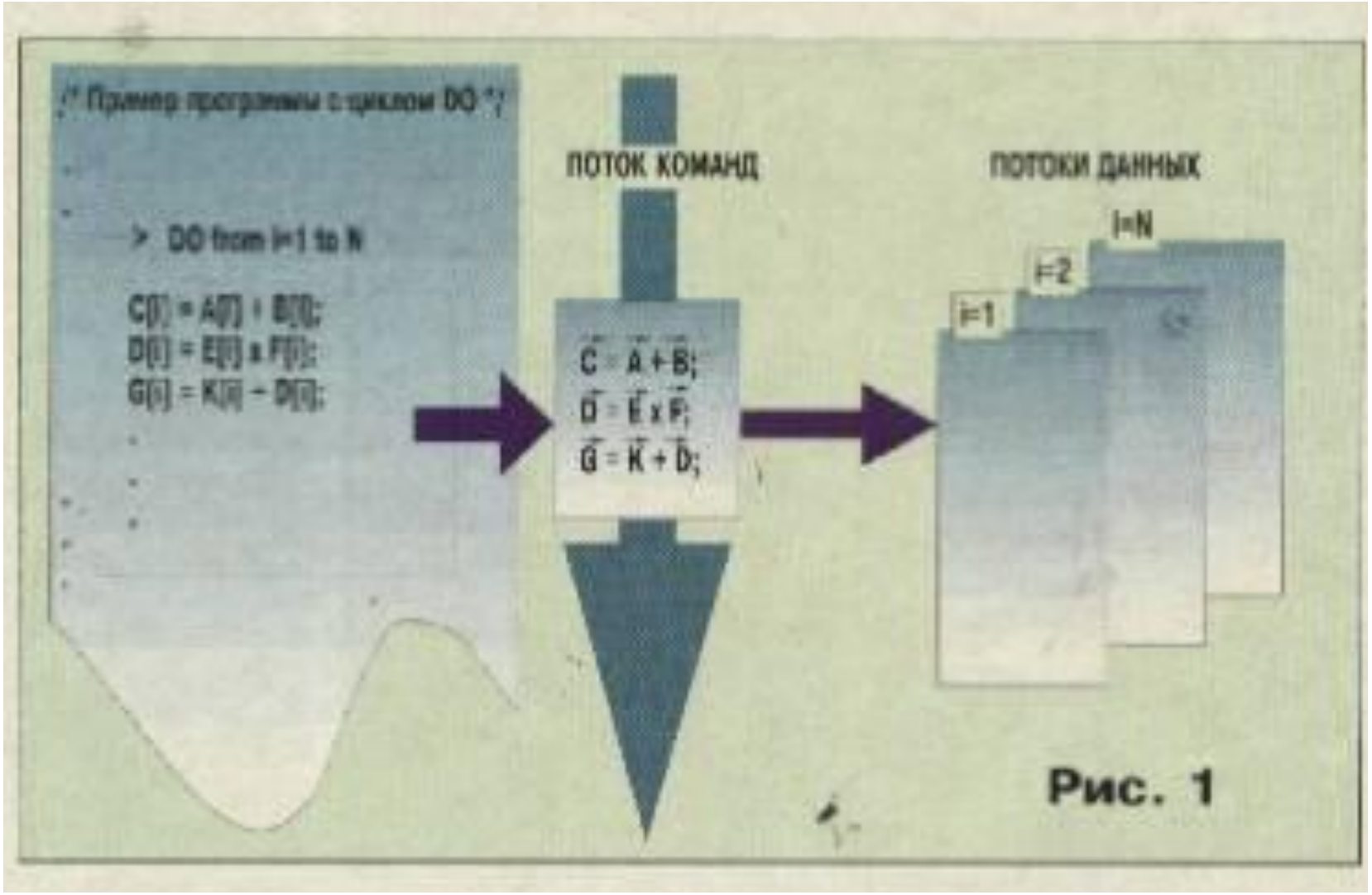
3*n операций (сложений, приращений индексов и условный переход).

Независимые данные. Если выполнять параллельно, за одну векторную операцию, то реализуются сразу две возможности ускорения вычислений: меньше команд объектного кода и одна операция сложения.

При аппаратном решении- другая архитектура. Один поток команд обрабатывая тело цикла, обрабатывает много потоков параллельных данных.

SIMD - «одиночный поток команд, множественный поток данных»;

Пример с циклом



Виды параллелизма

Рис. 2

Средства для реализации уровней параллелизма

1. Для нижнего уровня – конвейер фаз операций.
2. Для несвязанных операций - набор арифметических блоков процессора.
3. Векторные операции.
4. Мультипроцессирование.

Правило 20/80 – 20 команд – 80%вычислений.

RISC – процессор

Reduced Instruction Set Coumpting - Архитектура вычислений с сокращенным набором команд. RISC — архитектура процессора, в которой быстродействие увеличивается за счёт упрощения инструкций, чтобы их декодирование было более простым, а время выполнения — меньшим. В системах команд первых RISC-процессоров даже отсутствовали команды умножения и деления.

Операции «память – память», «регистр-регистр», «память – регистр», « регистр-память»

CISC – процессор: Complete Instruction Set Coumpting - Архитектура вычислений с полным набором команд

RISC-процессоры 3-го поколения

Основные особенности RISC-процессоров:

1. *Сокращенный набор команд* (от 80 до 150 команд).
2. Большинство команд выполняется за 1 такт.
3. Большое количество регистров общего назначения.
4. Наличие жестких многоступенчатых конвейеров.
5. Все команды имеют простой формат, и используются немногие способы адресации.
6. Наличие вместительной раздельной кэш-памяти.
7. Применение оптимизирующих компиляторов, которые анализируют исходный код и частично меняют порядок следования команд.

Самыми крупными разработчиками RISC-процессоров считаются Sun Microsystems (архитектура SPARC - Ultra SPARC), IBM (многокристальные процессоры Power, однокристальные PowerPC - PowerPC 620) и др.

Особенности всех RISC-процессоров

1. являются 64-х разрядными и суперскалярными (запускаются не менее 4-х команд за такт);
2. имеют встроенные конвейерные блоки арифметики с плавающей точкой;
3. имеют многоуровневую кэш-память. Большинство RISC-процессоров кэшируют предварительно дешифрованные команды;
4. изготавливаются по КМОП-технологии с 4 слоями металлизации.

Примечание:

1. Для обработки данных применяется алгоритм динамического прогнозирования ветвлений и метод переназначения регистров, что позволяет реализовать внеочередное выполнение команд.
2. Повышение производительности RISC-процессоров достигается за счет повышения тактовой частоты и усложнения схемы кристалла

Суперскалярные процессоры

Суперскалярные процессоры – позволяющие выполнять 2 и более скалярных операций одновременно.

Под суперскалярностью подразумевается наличие более одного конвейера для обработки команд (в отличие от скалярной - одноконвейерной архитектуры). В МП Pentium команды распределяются по двум независимым исполнительным конвейерам (U и V).

Конвейер U может выполнять любые команды семейства IA-32, включая целочисленные команды и команды с плавающей точкой.

Конвейер V предназначен для выполнения простых целочисленных команд и некоторых команд с плавающей точкой.

Конвейер

Идея ***конвейерной обработки*** заключается в выделении отдельных этапов выполнения общей операции, причем так, чтобы каждый этап, выполнив свою работу, передавал бы результат следующему, одновременно принимая новую порцию входных данных. Выигрыш в скорости обработки данных получается за счет совмещения прежде разнесенных во времени операций.

Пятиступенчатый конвейер для обработки команд:

1. IF (Instruction Fetch) – считывание команды в процессор;
2. ID (Instruction Decode) - декодирование команды;
3. OP (Operaand Reading) – считывание операндов;
4. EX (Executing) - выполнение команды в АЛУ и доступ к кэш-памяти;
5. WB (Write Back) - обратная запись.

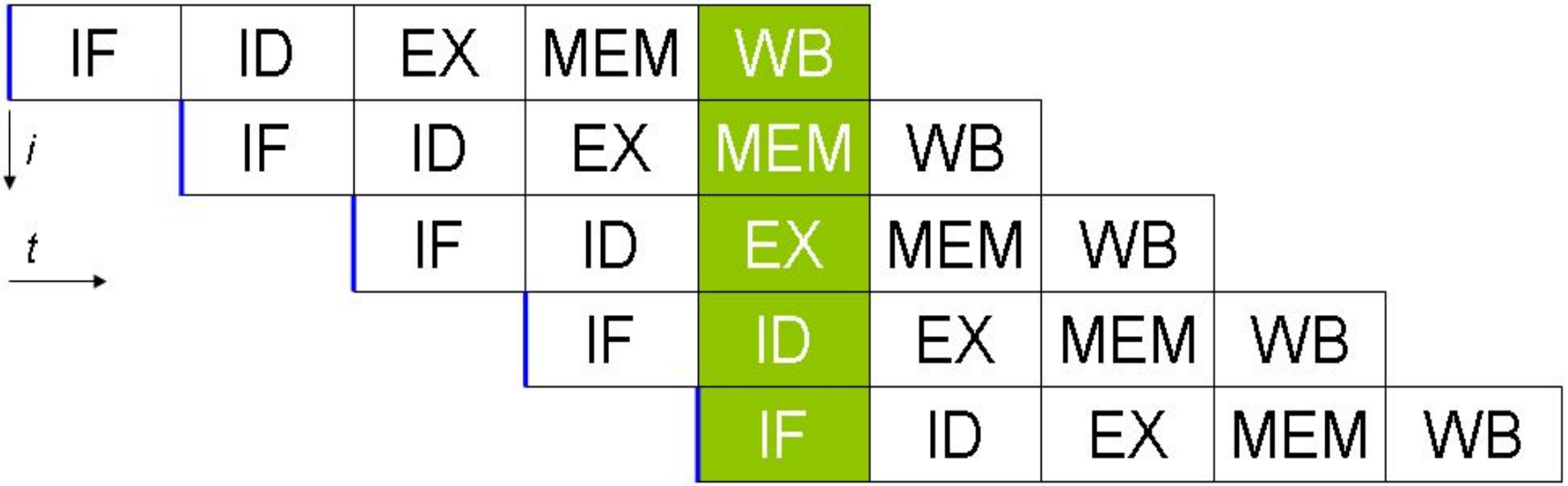
По сравнению с предыдущими поколениями IA-32, Pentium 4 содержит самый длинный конвейер команд, состоящий из 20 этапов и названный ***гиперконвейером***.

Заполнение конвейера

	1	2	3	4	5	6	7	8	9
i	IF	ID	OR	EX	WB				
i+1		IF	ID	OR	EX	WB			
i+2			IF	ID	OR	EX	WB		

Конвейер

Идея заключается в разделении обработки компьютерной инструкции на последовательность независимых стадий с сохранением результатов в конце каждой стадии. Это позволяет управляющим цепям процессора получать инструкции со скоростью самой медленной стадии обработки, однако при этом намного быстрее, чем при выполнении эксклюзивной полной обработки каждой инструкции от начала до конца.



Конвейер

Процессоры с конвейером внутри устроены так, что обработка инструкций разделена на последовательность стадий, предполагая одновременную обработку нескольких инструкций на разных стадиях. Результаты работы каждой из стадий передаются через ячейки памяти на следующую стадию, и так — до тех пор, пока инструкция не будет выполнена.

Подобная организация процессора, при некотором увеличении среднего времени выполнения каждой инструкции, тем не менее обеспечивает значительный рост производительности за счёт высокой частоты завершения выполнения инструкций.

Пример работы конвейера. Команда – 5 стадий

Пусть время выполнения одной стадии – в условных единицах.

20 ед. IF (Instruction Fetch) – считывание команды в процессор;

15 ед. ID (Instruction Decode) - декодирование команды;

20 ед. OR (Operaand Reading) – считывание операндов;

25 ед. EX (Executing) - выполнение команды в АЛУ и доступ к кэш-памяти;

20 ед. WB (Write Back) - обратная запись.

Итого 100 ед.

При последовательной обработке $T_{посл} = N*100$

При конвейерной $T_{конв} = 5*T+(N-1)*T$,

T – время такта = $макс(T_{if}, T_{id}, T_{or}, T_{ex}, T_{wb}) + dt$

dt = 5 ед.

Сравнение времени выполнения программы, содержащей различное количество команд (в условных единицах)

Кол-во команд	Последовател.	Конвейер
1	100	150
2	200	240
10	1000	420
100	10000	3120
1000	100000	30120

Преимущества и недостатки конвейера

Конвейер помогает не во всех случаях. Существует несколько возможных минусов. Конвейер инструкций можно назвать "полностью конвейерным", если он может принимать новую инструкцию каждый машинный циклКонвейер помогает не во всех случаях. Существует несколько возможных минусов. Конвейер инструкций можно назвать "полностью конвейерным", если он может принимать новую инструкцию каждый машинный цикл (англ. *en:clock cycle*). Иначе в конвейер должны быть вынужденно вставлены задержки, которые выравнивают конвейер, при этом ухудшается его производительность.

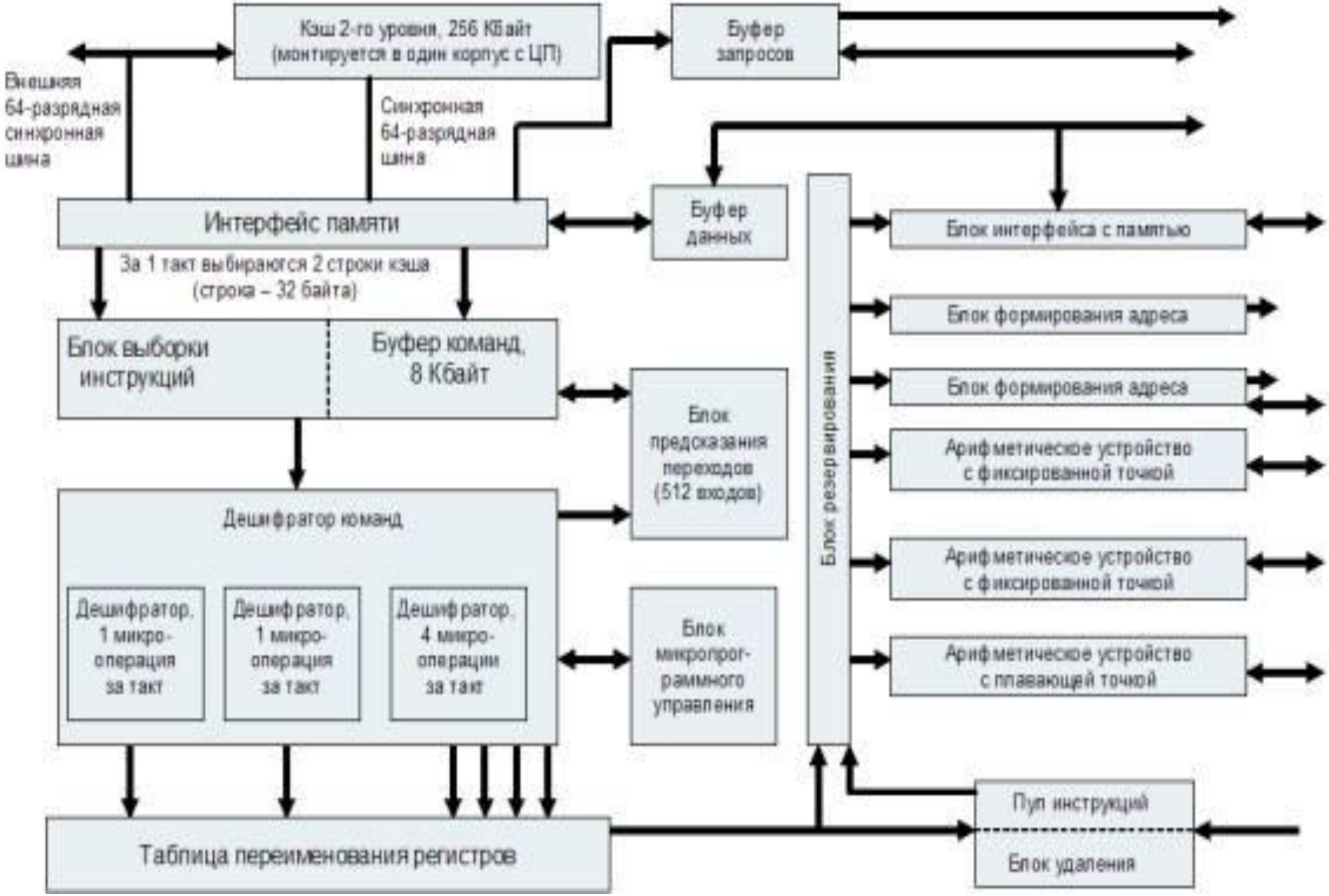
***Преимущества конвейера**:*

1. Время цикла процессора уменьшается, таким образом увеличивая скорость обработки инструкций в большинстве случаев.
2. Некоторые комбинационные логические элементы, такие как сумматоры (англ. *adders*) или умножители (англ. *multipliers*) могут быть ускорены путем увеличения количества логических элементов.

Недостатки конвейера:

- Бесконвейерный процессор исполняет только одну инструкцию за раз. Это предотвращает задержки веток инструкций (фактически, каждая ветка задерживается), и проблемы, связанные с последовательными инструкциями, которые исполняются параллельно. Следовательно, схема такого процессора проще и он дешевле для изготовления.
- Задержка инструкций в бесконвейерном процессоре слегка ниже, чем в конвейерном эквиваленте. Это происходит из-за того, что в конвейерный процессор должны быть добавлены дополнительные триггеры.
- У бесконвейерного процессора скорость обработки инструкций стабильна. Производительность конвейерного процессора предсказать намного сложнее, и она может значительно различаться в разных программах.

Блок схема микропроцессора Pentium Pro



Блок схема микропроцессора Pentium Pro

Одной из главных особенностей шестого поколения микропроцессоров архитектуры IA32 является *динамическое* (спекулятивное) *исполнение*. Под этим термином подразумевается следующая совокупность возможностей:

- *Глубокое предсказание ветвлений* (с вероятностью >90% можно предсказать 1015 ближайших переходов).
- *Анализ потока данных* (на 20-30 шагов вперед просмотреть программу и определить зависимость команд по данным или ресурсам).
- *Опережающее исполнение команд* (МП P6 может выполнять команды в порядке, отличном от их следования в программе).

Продолжение 1

Внутренняя организация МП P6 соответствует архитектуре RISC, поэтому блок выборки команд, считав поток инструкций IA-32 из L1 кэша инструкций, декодирует их в серию микроопераций. Поток микроопераций попадает в буфер переупорядочивания (пул инструкций). В нем содержатся как не выполненные пока микрооперации, так и уже выполненные, но еще не повлиявшие на состояние процессора. Для декодирования инструкций предназначены три параллельных дешифратора: два для простых и один для сложных инструкций. Каждая инструкция IA-32 декодируется в 1-4 микрооперации. Микрооперации выполняются пятью параллельными исполнительными устройствами: два для целочисленной арифметики, два для вещественной арифметики и блок интерфейса с памятью. Таким образом, возможно выполнение до пяти микроопераций за такт.

Продолжение 2

- Блок исполнительных устройств способен выбирать инструкции из пула в любом порядке. При этом благодаря блоку предсказания ветвлений возможно выполнение инструкций, следующих за условными переходами. Блок резервирования постоянно отслеживает в пуле инструкций те микрооперации, которые готовы к исполнению (исходные данные не зависят от результата других невыполненных инструкций) и направляет их на свободное исполнительное устройство соответствующего типа.
- Одно из целочисленных исполнительных устройств дополнительно занимается проверкой правильности предсказания переходов. При обнаружении неправильно предсказанного перехода все микрооперации, следующие за переходом, удаляются из пула и производится заполнение конвейера команд инструкциями по новому адресу.

Блок схема микропроцессора Pentium Pro
Продолжение 3

Взаимная зависимость команд от значения регистров архитектуры IA-32 может требовать ожидания освобождения регистров. Для решения этой проблемы предназначены 40 внутренних регистров общего назначения, используемых в реальных вычислениях.

Блок удаления отслеживает результат спекулятивно выполненных микроопераций. Если микрооперация более не зависит от других микроопераций, ее результат переносится на состояние процессора, и она удаляется из буфера переупорядочивания.

Блок удаления подтверждает выполнение инструкций (до трех микроопераций за такт) в порядке их следования в программе, принимая во внимание прерывания, исключения, точки останова и промахи предсказания переходов.

Дальнейшие усовершенствования

- *Динамическое исполнение команд* предполагает, что команды, не зависящие от результатов предыдущих операций, могут выполняться в измененном порядке (последующие раньше предыдущих), однако последовательность обмена с внешними устройствами (памятью и устройствами ввода/вывода) будет соответствовать программе. То есть процессор сам выбирает удобный ему порядок выполнения команд. Это позволяет повысить производительность процессора без увеличения тактовой частоты.

Дальнейшие усовершенствования

- *Архитектура двойной независимой шины* повышает суммарную пропускную способность. Одна шина (системная) служит для обмена с основной памятью и устройствами ввода/вывода, а другая (локальная) предназначена только для обмена с вторичным *кэшем*
- В процессор введен *кэш* второго уровня объемом 256—512 Кбайт.
- Возможно построение многопроцессорных систем (до четырех микропроцессоров)

Технология многоядерных чипов

По достижении температуры около 85° С значительно повышается вероятность нестабильной работы полупроводниковых компонентов, в том числе процессоров. Чтобы преодолеть этот лимит, исследователями из Федеральной политехнической школы Лозанны (Ecole Polytechnique Fédérale de Lausanne, EPFL) в сотрудничестве с IBM предложено в рамках проекта CMOSAIC решение с использованием технологии многоядерных чипов. Большинство из сегодняшних ПК имеют гордое обозначение "двухъядерный" или "четырёхъядерный“ и более. Тем не менее, в своё время наращивание производительности путём увеличения количества ядер на кристалле столкнётся с теми же ограничениями, что характерны для повышения степени интеграции одного ядра.

Многоядерный процессор

Многоядерный процессор — *центральный процессор*, содержащий два и более вычислительных *ядра* на одном процессорном кристалле или в одном корпусе.

В приложениях, оптимизированных под многопоточность, наблюдается прирост производительности на многоядерном процессоре. Однако, если приложение не оптимизировано, то оно не будет получать практически никакой выгоды от дополнительных ядер, а может даже выполняться медленнее, чем на процессоре с меньшим количеством ядер, но большей тактовой частотой.

Это в основном старые приложения, либо приложения, которым многопоточность не нужна (например, проигрыватель музыки) или невозможна.

Многоядерный процессор

Увеличение числа ядер процессоров признано как одно из приоритетных направлений увеличения производительности. Уже освоено производство 6-ти и более ядерных процессоров для домашних компьютеров, и 8-ми и 12-ти ядерных для серверных систем.

Процессор Montecito

В 2006 г. Был выпущен процессор Montecito, изготавливаемый по 90-нм техпроцессу, имел по сравнению с предшественником на 130-нм ядре Madison ряд других преимуществ: наличие Hyper-Threading (то есть он виден в системе как 4 логических процессора), заметно меньшее энергопотребление, более высокую производительность (в 1,5 раза и выше), вчетверо больший размер кэш-памяти (свыше 24 Мбайт: 2x1 Мбайт L2 инструкций, 2x12 Мбайт L3 данных), 1,72 миллиарда транзисторов против 410 миллионов и другое.

Микроархитектура Intel® нового поколения

Микроархитектура Intel нового поколения (кодовое название Nehalem) представляет собой следующий шаг в повышении энергоэкономичности, производительности и динамической масштабируемости процессоров.

Теперь производительность действительно стала доступна по запросу.

Динамическая масштабируемость и управление ядрами процессора, вычислительными потоками, кэш-памятью, интерфейсами и питанием обеспечивает энергоэкономичную производительность по требованию.

Микроархитектура Intel® нового поколения

Масштабирование производительности серверов, рабочих станций, ПК и мобильных устройств с поддержкой 2-8 и более ядер и до 16 и более потоков с технологией одновременной многопоточности (SMT), а также масштабирование размера кэш-памяти, системных интерфейсов и встроенных контроллеров памяти.

Одновременная многопоточность повышает производительность массовых вычислительных систем с 1-16 и большим количеством потоков, оптимизированных для архитектуры многоядерных процессоров нового поколения.

Масштабируемая общая память в технологии Intel® QuickPath обеспечивает распределение ресурсов памяти между процессорами с помощью встроенных контроллеров памяти и высокоскоростных интерфейсов, позволяя полностью раскрыть преимущества производительности многоядерных процессоров Intel®.

Многоуровневая общая кэш-память повышает производительность и эффективность работы, снижая задержки доступа к часто используемым данным.

Двухядерный процессор

Классическая двухпроцессорная система (самый простой вариант многопроцессорности) подразумевает наличие двух отдельных процессоров. Оба процессора одинаковы и равноправны (если речь идет о симметричной многопроцессорной системе, SMP), оба используют общую системную шину, но управляют шиной по очереди. Чередование осуществляет специальный блок - арбитр шины, передающий управление тому или другому процессору по определенным правилам.

Оба процессора равноправно разделяют доступ к общим системным ресурсам - оперативной памяти, периферии и т. д. Вычислительная нагрузка между процессорами распределяется средствами операционной системы, вот почему выигрыш в производительности от применения SMP возможен только при использовании оптимизированных под многопоточность приложений.

Двухядерный процессор

У двухядерных процессоров на одном кристалле, рядом друг с другом, расположены два независимых процессорных ядра. Площадь кристалла у двухядерного процессора чуть меньше удвоенной площади одноядерного (для процессоров Intel - 206 и 112 кв. мм соответственно), число транзисторов также отличается почти вдвое (230 и 125 млн). Двухядерный кристалл упакован в стандартный корпус (LGA 775 у Intel, Socket 939 у AMD), для охлаждения используется один кулер, так что системные платы для двухядерных процессоров вполне умещаются в стандартный форм-фактор ATX.

Однако имеется некоторая специфика, связанная с питанием и охлаждением.

Двухядерный процессор

В случае двухпроцессорной или двухядерной системы присутствует следующая картина. Если вычислительных потоков два, каждый из них получает в свое распоряжение одно из ядер целиком, безраздельно. Потоки могут обрабатываться непрерывно, даже если выполняемые приложения вовсе не умеют распараллеливаться. Кажется, что тут уж возможно получить двукратный выигрыш в производительности, однако реально этого не происходит.

Двухядерный процессор

Причина в том, что часть аппаратных ресурсов вычислительной системы оба ядра используют совместно (например, оперативную память, периферию), так что некоторое пересечение интересов будет всегда. Тем не менее, для двухядерной системы повышение производительности на 80-90 % - вещь вполне достижимая, зафиксированная в тестировании на реальных приложениях.

Двухядерный процессор и приложения

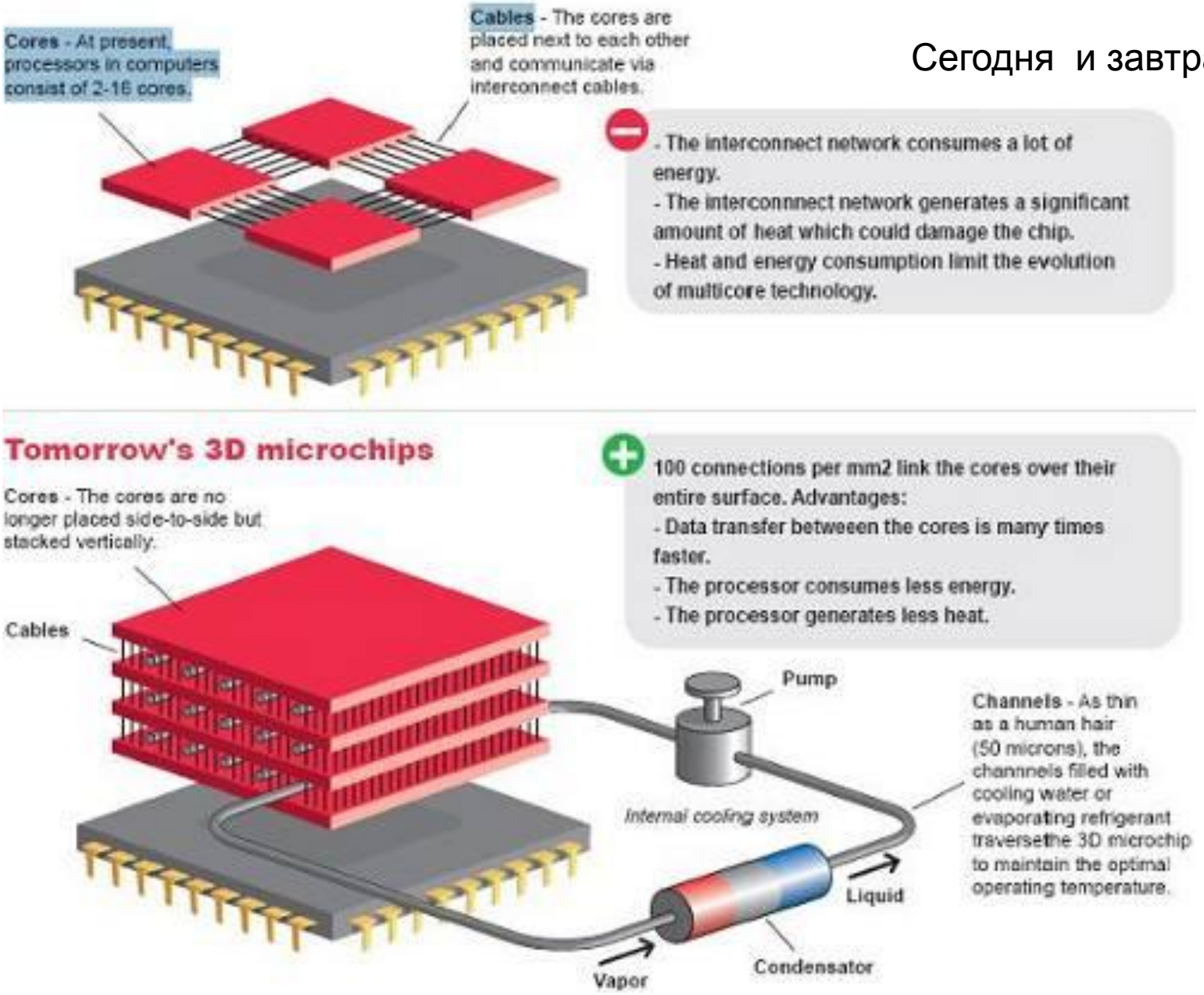
Конечно, конкретный прирост производительности существенным образом зависит от того, сколько приложений и какие именно приложения запущены в одновременную работу, умеют ли эти приложения распараллеливаться на несколько потоков, и так далее. Реальный, измеряемый, заметный пользователю выигрыш получается в двух случаях:

1. выполняется только одно приложение, которое рассчитано на параллельные вычисления. К таковым приложениям относятся программы для кодирования и редактирования аудио- и видеопотоков, обработки изображений (Adobe Premier, Windows Media Encoder, Adobe Photoshop), программные пакеты 3D-моделирования и рендеринга (3D Studio Max, LightWave), профессиональные пакеты САПР. Именно для таких приложений время выполнения одиночной задачи сокращается в полтора-два раза.

Двухядерный процессор и приложения

2. необходимость одновременного выполнения хотя бы двух разных приложений. В этом случае, даже если каждое приложение создает только один поток, выигрыш в производительности все равно будет существенным.

Сегодня и завтра



Трёхмерные процессоры

Трёхмерные процессоры основаны на идее многоядерных чипов. Но размещаются они иным способом – вертикально, а не бок о бок.

Преимущество в том, что вся поверхность одного вычислительного элемента может быть подключена к следующему слою с количеством соединений до 100 тыс. на мм^2.

Множество коротких проводников приведут к повышению пропускной способности между ядрами, снижению энергопотребления и тепловыделения.

Новейший 3,5-дюймовый SBC (Session Border Controller) **IB953**

IBASE Technology Inc., ведущий поставщик промышленных материнских плат и встраиваемых систем, запустил свой новейший 3,5-дюймовый SBC (Session Border Controller) **IB953**, который оснащен разъемом M.2 3052, совместимым с 5G, и новейшим процессором Intel Core i3/i5/i7 11-го поколения, построенном по 10 нм техпроцессу SuperFin. Все это обеспечивает 25% прирост производительности и скорости.

Например: процессор Intel Core i7-1185G7 включает новую графическую архитектуру Iris Xe (Xe-LP), которая работает на частоте 1,35 ГГц, что на 250 МГц больше, чем у предыдущей графики. Это важно для большей энергоэффективности и возможности одновременно управлять четырьмя дисплеями 4K HDR через два порта DisplayPort, eDP и LVDS.

IB953

Для обеспечения надежной обработки данных, IB953 располагает портами: 3x USB 2.0, 3x USB 3.1, 2x COM, 3x M.2 (поддерживаются B-Key и M-Key, NVMe и CNVi) и 2x GbE.

Кроме того, одноплатный компьютер использует оперативную память DDR4-3200 SO-DIMM объемом до 64 ГБ, поддерживает функцию энергосбережения ErP / EuP для снижения энергопотребления материнской платы в выключенном состоянии и имеет широкодиапазонный вход питания 12 ~ 24 В постоянного тока, подходящий для различных промышленных приложений.

Поддерживаемые операционные системы: 64-разрядные Windows 10 и Linux Ubuntu.

При своих компактных размерах (всего 146 x 102 мм) одноплатный компьютер обеспечивает впечатляющую производительность, разнообразие портов ввода-вывода, что делает его идеальной платформой для широкого спектра требовательных приложений в областях автоматизации производства, машинного зрения, здравоохранения, а также подходит для розничной торговли.