Методы повышения производительности микропроцессора

Повышение производительности вычислительно системы

Основой для повышения производительности вычислительной системы является принцип "**совмещения операций**", при котором аппаратура компьютера в любой момент времени выполняет одновременно более одной базовой операции.
Принцип совмещения включает в себя два понятия: **параллелизм** и **конвейеризацию**.

Параллелизм (или совмещение операций) достигается путем воспроизведения вычислительного процесса в нескольких копиях аппаратной структуры. Высокая производительность достигается за счет одновременной работы всех элементов структур, осуществляющих решение различных частей задачи.

Конвейеризация (или конвейерная обработка) в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые *ступенями*, и выделении для каждой из них отдельного блока вычислительной аппаратуры.

При этом конвейерную обработку можно использовать для совмещения этапов выполнения разных команд.
Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд.

Конвейеризация эффективна только тогда, когда загрузка конвейера близка к полной, а скорость подачи новых операндов соответствует максимальной производительности конвейера. Если происходит задержка, то параллельно будет

Цикл выполнения команды

Программа состоит из машинных команд. Программа загружается в память компьютера. Затем программа начинает выполняться, то есть процессор выполняет машинные команды в той последовательности, в какой они записаны в программе (исполняет алгоритм).

Цикл выполнения команды – это последовательность действий, которая совершается процессором при выполнении одной машинной команды. При выполнении каждой машинной команды процессор должен выполнить как минимум три действия: *выборку*, *декодирование* и *выполнение*. Если в команде используется операнд, расположенный в оперативной памяти, то процессору придётся выполнить ещё две операции: *выборку операнда из памяти* и *запись результата в память*.

Для того чтобы процессор знал, какую команду нужно выполнять в определённый момент, существует **счётчик команд** – специальный регистр, в котором хранится адрес команды, которая должна быть выполнена после выполнения текущей команды. То есть при запуске программы в этом регистре хранится адрес первой команды.

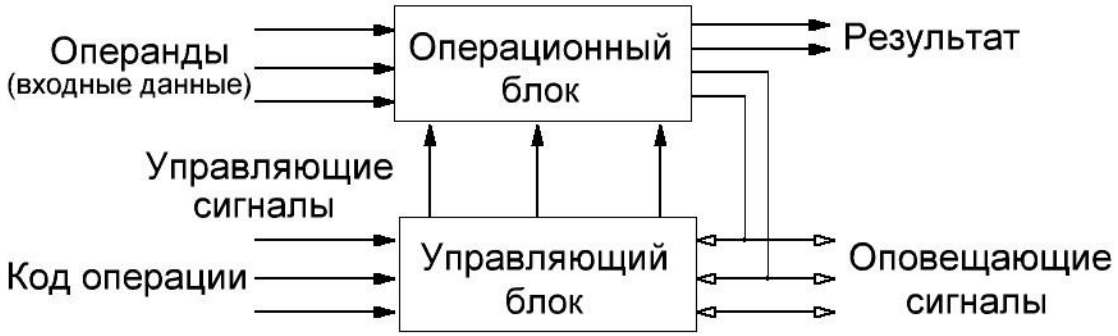
Счётчик команд работает со сверхоперативной памятью, которая находится внутри процессора. Эта память носит название **очередь команд**, куда помещается одна или несколько команд непосредственно перед их выполнением. То есть в счётчике команд хранится адрес команды находящейся в очереди команд, а не адрес команды в оперативной памяти.

Операции исполняемые при выполнении команды

- **Выборка команды**. Блок управления извлекает команду из *памяти программ*, копирует её во *внутреннюю память процессора* и увеличивает значение *счётчика команд* на длину этой команды (разные команды могут иметь разный размер).
- **Декодирование команды**. Блок управления определяет тип выполняемой команды, пересылает указанные в ней операнды в АЛУ и генерирует электрические сигналы управления АЛУ, которые соответствуют типу выполняемой операции.
- **Выборка операндов**. Если в команде используется операнд, расположенный в оперативной памяти, то блок управления начинает операцию по его выборке из памяти.
- **Выполнение команды**. АЛУ выполняет указанную в команде операцию, сохраняет полученный результат в заданном месте (регистр процессора) и обновляет состояние флагов (регистр флагов), по значению которых программа может судить о результате выполнения команды.
- **Запись результата в память**. Если результат выполнения команды должен быть сохранён в памяти, блок управления начинает операцию сохранения данных в памяти

Конвейеризация

Декомпозиция вычислительного устройства



Совмещение операций во времени с использованием **конвейерного регистра**

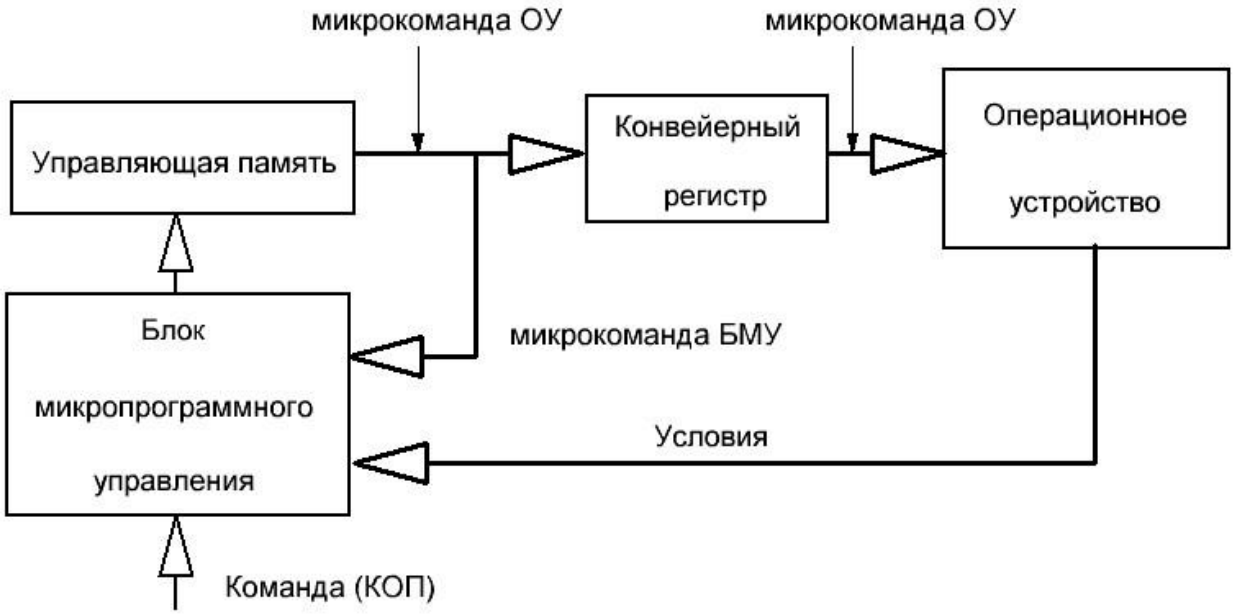


Рисунок 1 – Структура процессора с конвейерным регистром

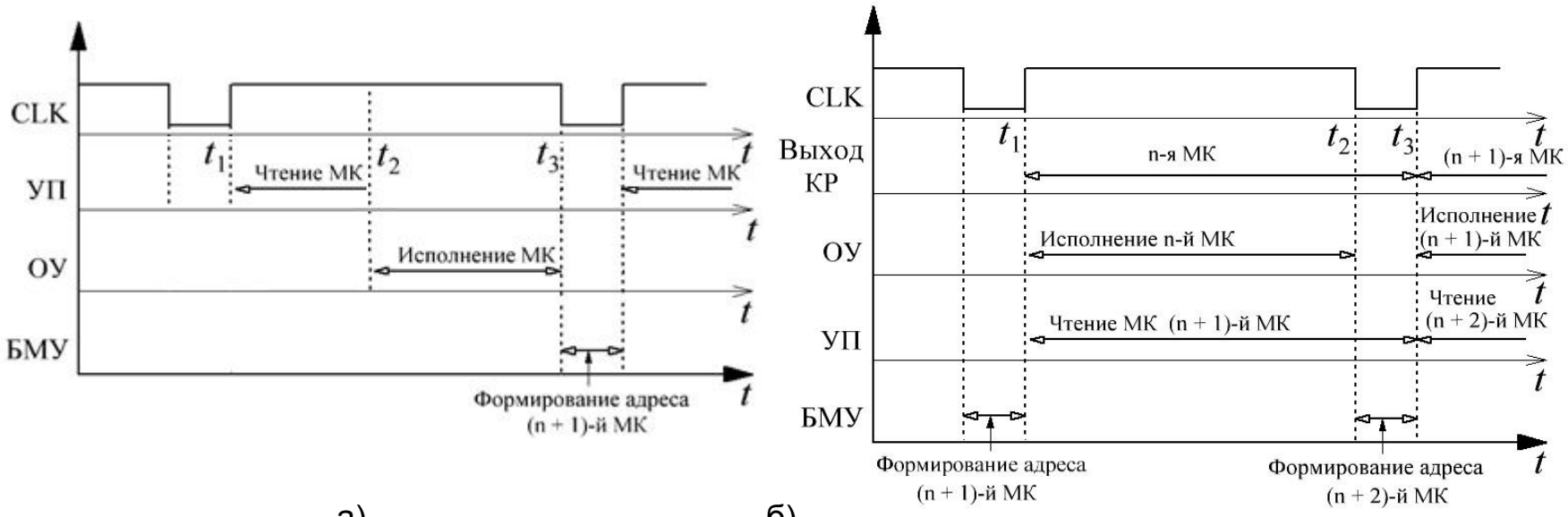


Рисунок 2 – Временная диаграмма работы процессора, обычного (а) и с конвейерным регистром (б)

Пока **конвейерный регистр** (КР) хранит текущую микрокоманду, под действием которой в **операционном устройстве** (ОУ) выполняется операция, из **регистра адреса** (РА) микрокоманд **блока микропрограммного управления** (БМУ) выдаётся адрес следующей микрокоманды, и в **управляющей памяти** (УП) протекают процессы, связанные с **выборкой**, в то же время в **логической схеме определения следующего адреса** (ЛСх) **формируется адрес** очередной микрокоманды.

Пример реализации конвейера команд в микропроцессорах

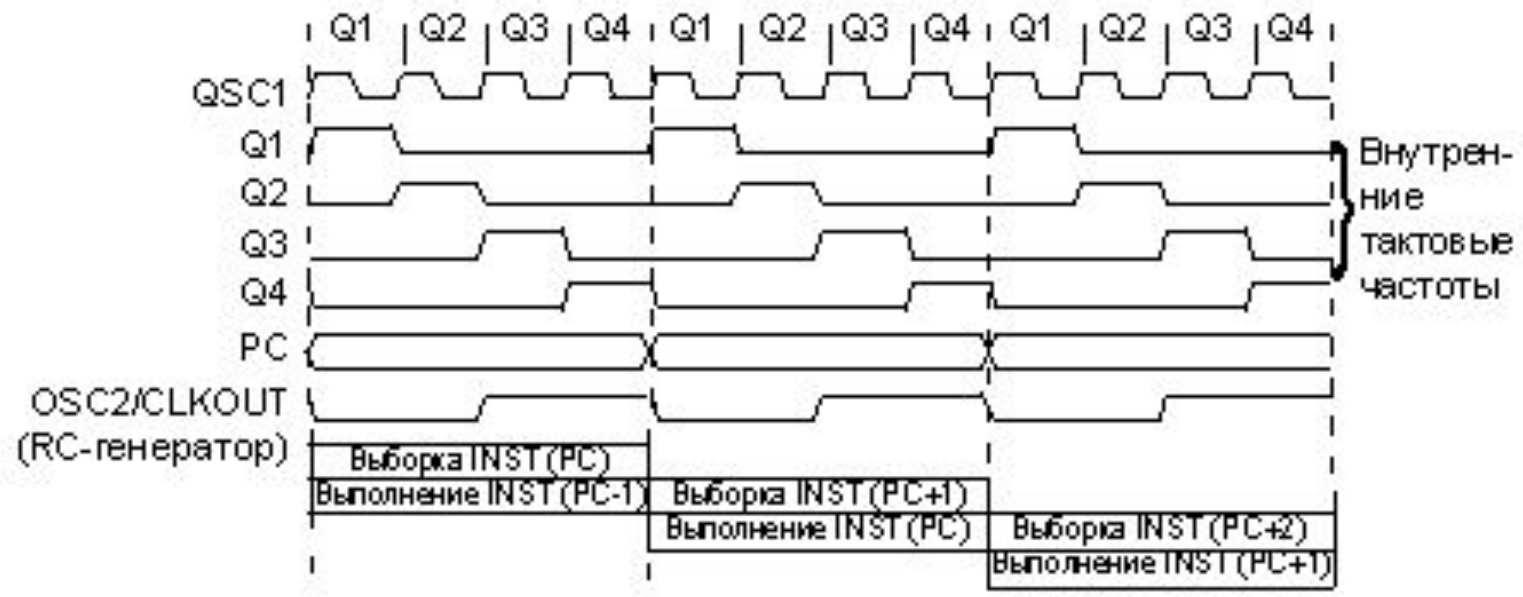


Рисунок 3 – Схема тактирования и выполнения команды в конвейере PIC

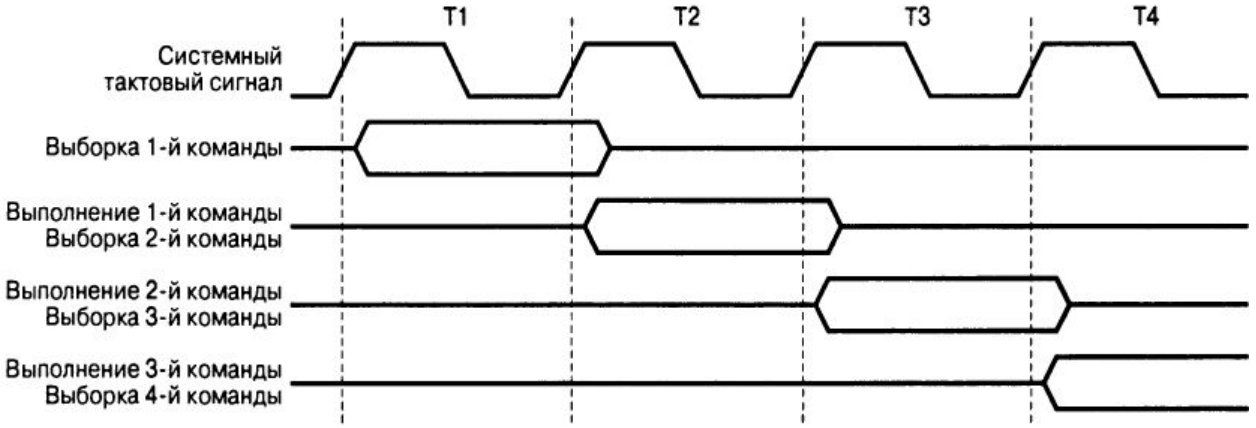


Рисунок 4 – Последовательность выполнения команд в конвейере AVR

Очередь команд

Очередь команд – позволяет выбирать (аккумулировать) несколько последующих команд, пока выполняется предыдущая.
Физически очередь команд представляет собой регистровый файл (несколько регистров организованных по принципу **FIFO**), которые содержат последовательность из нескольких команд следующих за исполняемой. Команды из очереди считываются в том же порядке, что и записываются.

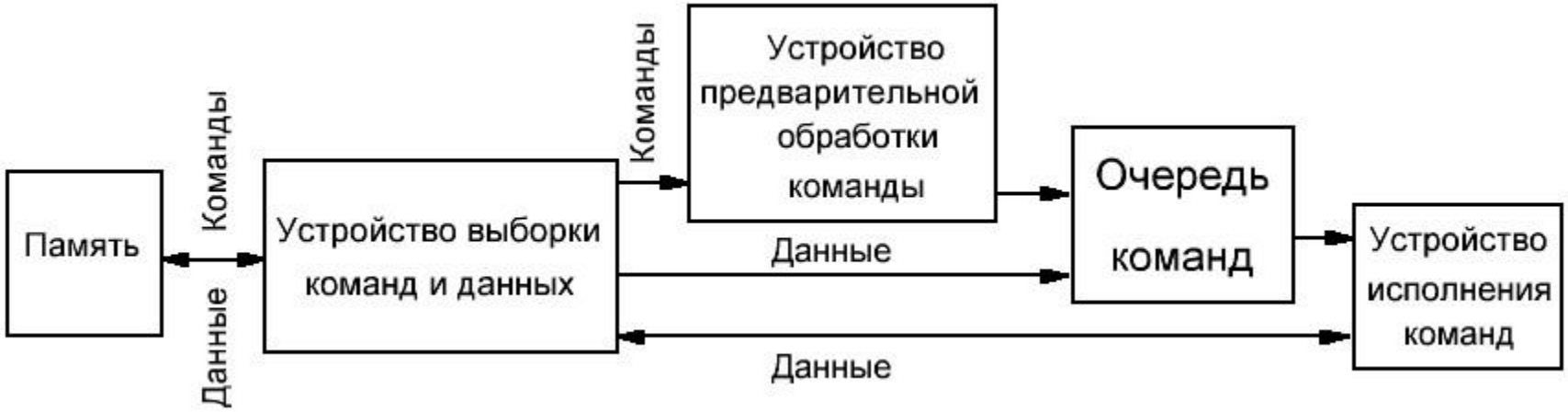


Рисунок 5 – Структура конвейерного микропроцессора с очередью команд

Достоинство: сокращает время простоя устройства исполнения команд – операционное устройство (ОУ).
Недостаток: При переходах на удаленную ячейку памяти конвейер сбрасывается. Это замедляет работу процессора. Для уменьшения влияния этого недостатка **программы должны быть специальным образом оптимизированы** под использование конвейера команд.

Шинный интерфейс инициирует выборку из памяти следующего командного слова, когда в очереди оказываются два свободных (пустых) байта. В большинстве случаев очередь команд содержит минимум один байт потока команд, и операционное устройство не ожидает выборки команды из памяти.

Очередь обеспечивает положительный эффект при естественном порядке выполнения команд. Когда операционное устройство выполняет команду передачи управления, шинный интерфейс сбрасывает очередь, выбирает команду по новому адресу, передает ее в операционное устройство, а затем начинает заполнение очереди из следующих ячеек. Эти действия не выполняются при условных и безусловных переходах, вызовах подпрограмм, возвратах из подпрограмм и при обработке прерываний. Шинный интерфейс приостанавливает выборку команд, когда операционное устройство запрашивает операцию считывания или записи в память или порт ввода-вывода.

При готовности операционного устройства выполнять команду оно считывает команду (КОП) из очереди команд, а затем выполняет предписанную командой операцию.

При многобайтных командах из очереди считываются и другие байты команды.

Когда операционное устройство готово считать командный байт (КОП), а очередь команд пуста, то ему приходится ожидать выборки соответствующего командного слова из памяти программ, которую производит шинный интерфейс.

Если команда требует обращения к памяти или порту ввода-вывода, операционное устройство запрашивает шинный интерфейс на выполнение необходимого цикла шины. Когда шинный интерфейс не занят выборкой команды, он удовлетворяет запрос немедленно; в противном случае операционное

Очередь команд характеризуется:

1. **Разрядностью регистров – $n_{ОК}$**
2. **Длиной очереди (количество регистров) – $Q_{ОК}$**

$$Q_{ОК,опт} = \lambda_{ОК} \cdot \overline{t_{обр}}$$

где $\lambda_{ОК}$ – интенсивность поступления информации в ОК;
$t_{обр}$ – среднее время обработки информации в микропроцессоре.

$$\overline{t_{обр}} = \bar{t}_К - P_{КОН}\beta_П T_{эфф}$$

где $t_К$ – среднее время выполнения команд в конвейерном микропроцессоре с очередью команд (с учётом времени обращения к памяти);
$P_{КОН}$ – **вероятность межкомандных конфликтов** в конвейере, связанных с выполнением команд передачи управления, прерывания, запоминания;
$\beta_П$ – среднее число обращений к памяти при выполнении команды;
$T_{ЭФ}$ – эффективное время доступа к памяти.

Оценить интенсивность поступление команд можно исходя из вариантов размещения ОК:

$$\lambda_{ОК\,max} = \frac{n_П}{n_{ОК} T_{ЭФ}}$$

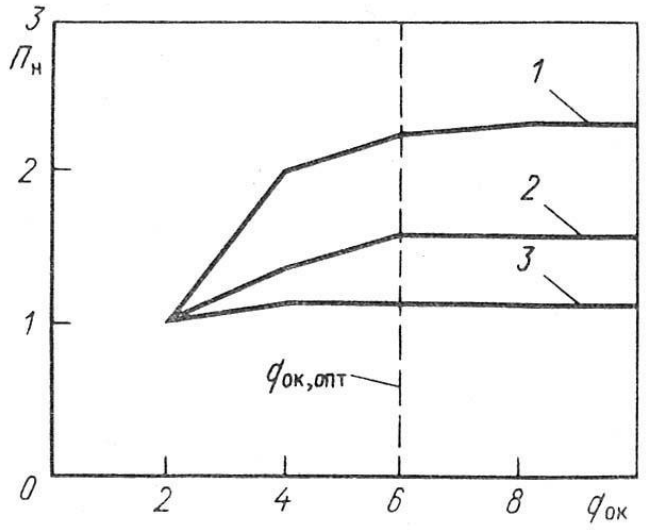
Пример. Работа очереди команд для микропроцессора i8086

$$\beta T_{ЭФ} = \frac{(П_{Н2} - П_{Н1})}{П_{Н1} П_{Н2}}$$

Пн1 – без конвейера
Пн2 – конвейер с 1 уровнем совмещения
Пн3 – Пн2 + ОК
Пн4 – Пн2 + 1 уровень совмещения + ОК

где $П_{Н1}$ = 1; $П_{Н2}$ = 1,35; $П_{Н3}$ = 1,5; $П_{н4}$ = 1,65
$n_{ОК}$ = 8 бит, т.к. 65% команд имеют длину 1 байт
$n_{П}$ = 16 бит

$$Q_{ОК\,опт} \approx \left\lceil \frac{16 \cdot 1{,}35}{8 \cdot (1{,}35 - 1) \cdot 1{,}5} \right\rceil = 6$$



1 – команды регистр-регистр
2 – равномерная смесь команд
3 – команды со сложными способами адресации

Рисунок 6 – Производительность, как функция длины ОК

Параллелизм в вычислительных системах

Классификация параллельных вычислительных систем

В 1966 году Майкл Флинн предложил следующую классификацию вычислительных систем, основанную на количестве потоков входных данных и количестве потоков команд, которые эти данные обрабатывают:

	Один поток инструкций	Несколько потоков инструкций
Один поток данных	SISD	MISD
Несколько потоков данных	SIMD	MIMD

Рисунок 7 – Классификация архитектур ЭВМ по Флинну

Вычислительный процесс типа SISD (англ. *Single Instruction Single Data*) – одиночный поток команд и одиночный поток данных.

Рисунок 8 – Структура вычислительного процесса SISD

К этому классу относятся последовательные компьютерные системы, которые имеют один центральный процессор, способный обрабатывать только один поток последовательно исполняемых инструкций.

В настоящее время практически все высокопроизводительные системы имеют более одного центрального процессора, однако каждый из них выполняет несвязанные потоки инструкций, что делает такие системы комплексами SISD-систем, действующих на разных пространствах данных.

Для увеличения скорости обработки команд и скорости выполнения арифметических операций может применяться конвейерная обработка.

Относится к фон-Неймановской архитектуре.

Вычислительный процесс **SIMD** (англ. *Single Instruction Multiple Data* — последовательный поток команд и параллельный поток данных), который предполагает распараллеливание обработки информации за счет увеличения числа операционных узлов требуемого типа.
Один поток инструкций выполняет вычисления одновременно с разными данными.

Рисунок 9 – Структура вычислительного процесса SIMD

Такие компьютеры называются векторными, так как подобные операции выполняются аналогично операциям с векторами (когда, например, сложение двух векторов означает одновременное сложение всех их компонентов).

Зачастую векторные инструкции присутствуют в дополнение к обычным «скалярным» инструкциям, и называются SIMD-расширением (или векторным расширением).

Технология **SIMD** впервые применена в процессоре **Intel486** **MMX** и рассчитана на мультимедийное, графическое и коммуникационное применение.

В систему команд процессора **Intel486** для поддержки технологии **MMX** (англ. *multimedia extension*) введено **57** **дополнительных команд** **для одновременной обработки нескольких единиц данных** (команды пересылки, арифметические, логические команды и команды преобразования форматов данных).

Команды MMX доступны из любого режима процессора.

Развитие технологии **SIMD** для чисел с плавающей запятой получило название **SSE** (***streaming SIMD Extention***), появившееся впервые в МП **Pentium III.**

Вычислительный процесс типа MISD (англ. *Multiple Instruction Single Data* — параллельный поток команд и последовательный поток данных). Определение подразумевает наличие в архитектуре многих процессоров, обрабатывающих один и тот же поток данных.

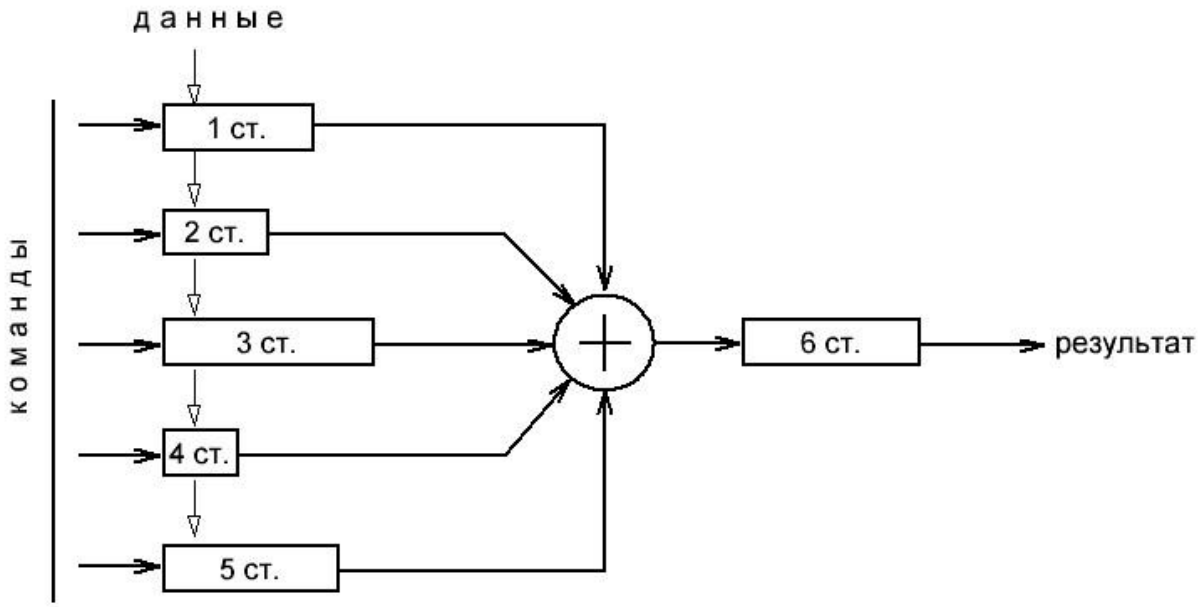


Рисунок 10 – Структура вычислительного процесса MISD

Вычислительный процесс **MISD обеспечивает большую скорость обработки информации** по сравнению с последовательным процессом типа **SIMD**, поскольку производительность конвейерных вычислительных средств будет определяться временем выполнения максимального по длительности этапа вычислений в соответствующей ступени вычислительного конвейера, а не полным временем выполнения команд и операций.

Вычислительный процесс типа MIMD (англ. *Multiple Instruction Multiple Data*) – множественный поток команд и множественный поток данных.

Разные потоки инструкций оперируют различными данными. Это системы наиболее общего вида, поэтому их проще всего использовать для решения различных параллельных задач.

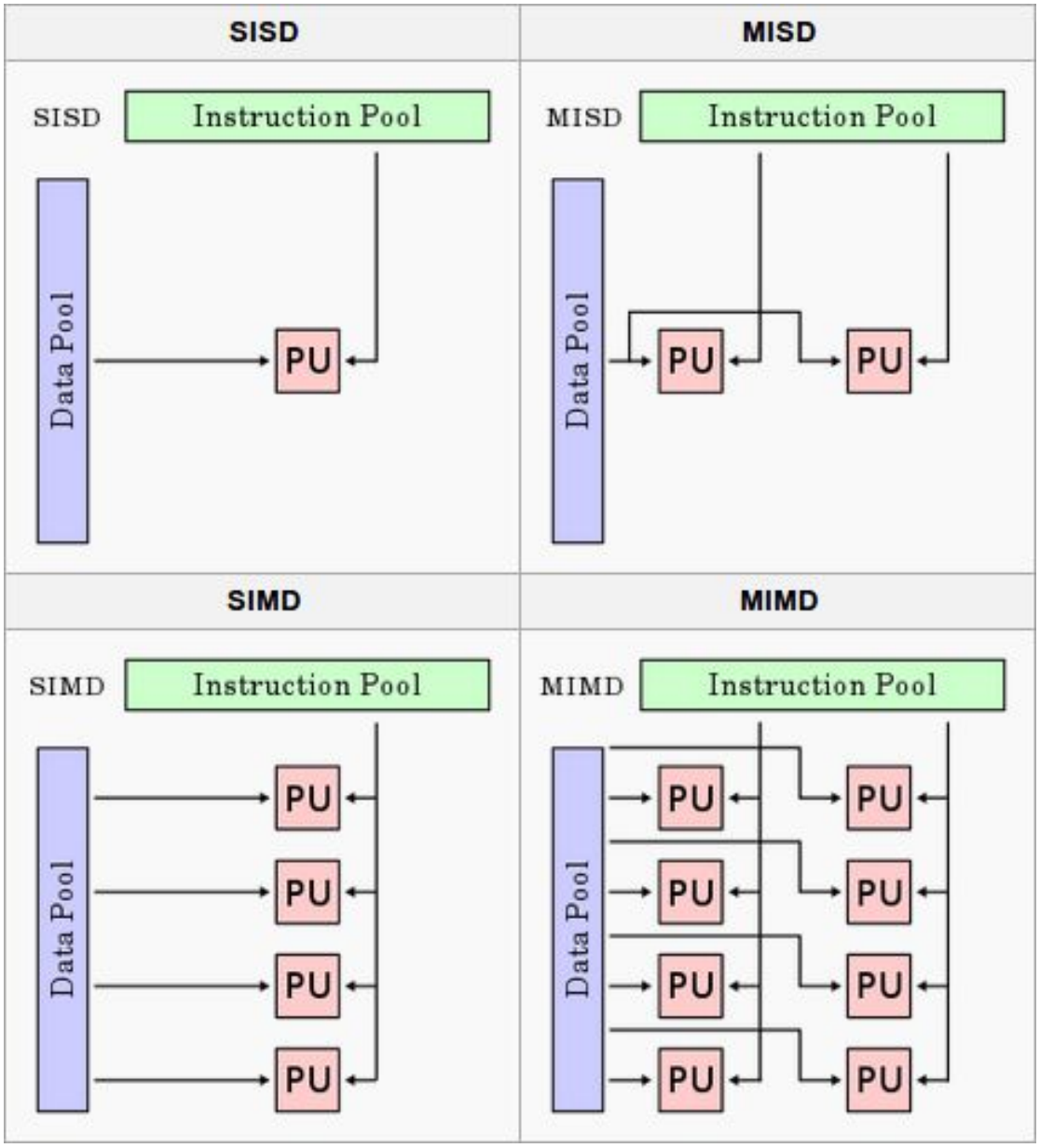
MIMD означает, что каждый процессор может обрабатывать отдельный поток инструкций над его собственными локальными данными.

В отличие от упомянутых выше многопроцессорных SISD-машин, команды и данные в MIMD связаны, потому что они представляют различные части одной и той же задачи.

Например, MIMD-системы могут параллельно выполнять множество подзадач с целью сокращения времени выполнения основной задачи.

MIMD-системы принято разделять (классификация Джонсона) на:

- **системы с общей памятью** (несколько вычислителей имеют общую память) — **архитектура SMP**;
- **системы с распределенной памятью** (каждый вычислитель имеет свою память; вычислители могут обмениваться данными) — **архитектура MPP**.
- **системы с неоднородным доступом к памяти** в которых доступ к памяти других вычислителей существует, но он значительно медленнее, чем до ступ к «своей» памяти – **архитектура NUMA**.


SISD
SISD
Instruction Pool
Data Pool
PU
MISD
MISD
Instruction Pool
Data Pool
PU
PU
SIMD
SIMD
Instruction Pool
Data Pool
PU
PU
PU
PU
MIMD
MIMD
Instruction Pool
Data Pool
PU
PU
PU
PU
PU
PU
PU
PU

SMP-архитектура

Симметричная многопроцессорная архитектура – **SMP** (англ. *symmetric multiprocessing*) архитектура.

Главной особенностью систем с архитектурой SMP является наличие общей физической памяти, разделяемой всеми процессорами.

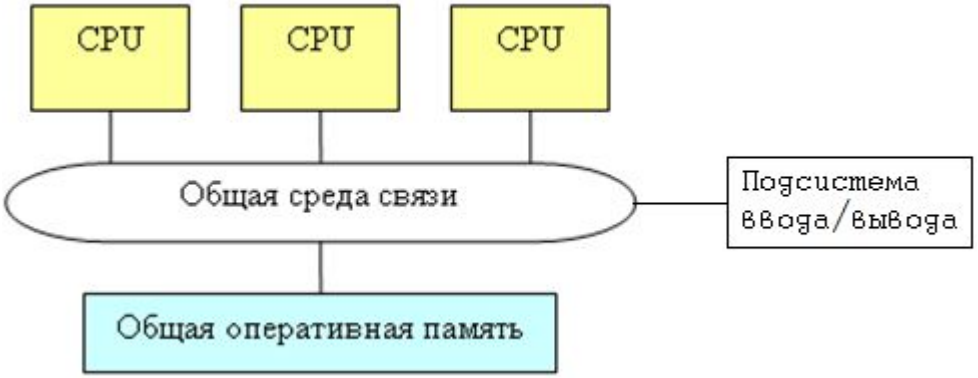


Рисунок 11 – SMP архитектура ЭВМ

Память в SMP системе служит, в частности, для передачи сообщений между процессорами, при этом все вычислительные устройства при обращении к ней имеют равные права и одну и ту же адресацию для всех ячеек памяти. Поэтому SMP-архитектура называется симметричной.

SMP системы позволяют любому процессору работать над любой задачей независимо от того, где в памяти хранятся данные для этой задачи.

С должной поддержкой операционной системы, SMP системы могут легко перемещать задачи между процессорами эффективно распределяя вычислительную нагрузку.

С другой стороны, память гораздо медленнее процессоров, которые к ней обращаются, даже однопроцессорным машинам приходится тратить значительное время на получение данных из памяти. В SMP только один процессор может обращаться к памяти в каждый момент времени.

Основные **преимущества** SMP-систем:

— простота и универсальность для программирования.

Архитектура SMP не накладывает ограничений на модель программирования, используемую при создании приложения: обычно используется модель параллельных ветвей, когда все процессоры работают независимо друг от друга. Однако можно реализовать и модели, использующие межпроцессорный обмен. Использование общей памяти увеличивает скорость такого обмена, пользователь также имеет доступ сразу ко всему объему памяти. Для SMP-систем существуют довольно эффективные средства автоматического распараллеливания;

— простота эксплуатации.

Как правило, SMP-системы используют систему кондиционирования, основанную на воздушном охлаждении, что облегчает их техническое обслуживание;

— относительно невысокая цена.

Недостатки: системы с общей памятью плохо масштабируются.

Причиной плохой масштабируемости является то, что в каждый момент времени системная магистраль способна обрабатывать только одну транзакцию, вследствие чего возникают проблемы разрешения конфликтов при одновременном обращении нескольких процессоров к одним и тем же областям общей физической памяти. Вычислительные элементы начинают друг другу мешать. Практически можно задействовать не более 32 процессоров, хотя в настоящее время в реальных системах конфликты могут происходить уже при наличии 8-24 процессоров.

Большинство современных вычислительных систем обычного применения строится на основе SMP архитектуры.

MPP-архитектура

Массивно-параллельная архитектура MPP (англ. *Massive Parallel Processing*,) — класс архитектур параллельных вычислительных систем.

Особенность архитектуры состоит в том, что:

— память физически разделена,

— система строится из отдельных модулей, содержащих процессор, локальный банк оперативной памяти, коммуникационные процессоры или сетевые адаптеры, иногда, жесткие диски и/или другие устройства ввода/вывода.

— доступ к банку операционной памяти из данного модуля имеют только процессоры из этого же модуля.

— модули соединяются специальными коммуникационными каналами.

В отличие от SMP-систем, в машинах с раздельной памятью каждый процессор имеет доступ только к своей локальной памяти, в связи с чем не возникает необходимости в потактовой синхронизации процессоров.

Рисунок 12 – MPP архитектура ЭВМ

Достоинства MPP: хорошая масштабируемость.

Легко подобрать оптимальную конфигурацию вычислительной системы изменяя количество процессоров если заранее известна требуемая вычислительная мощность.

Недостатки:

— отсутствие общей памяти заметно снижает скорость межпроцессорного обмена, поскольку нет общей среды для хранения данных, предназначенных для обмена между процессорами. Требуется специальная техника программирования для реализации обмена сообщениями между процессорами;

— каждый процессор может использовать только ограниченный объем локального банка памяти;

Вследствие указанных архитектурных недостатков требуются значительные усилия для того, чтобы максимально использовать системные ресурсы. Поэтому написать эффективную программу для компьютеров с MPP архитектурой очень сложно, а для некоторых алгоритмов – иногда просто невозможно. Именно этим определяется высокая цена программного обеспечения для массивно-параллельных систем с раздельной памятью.

Наиболее производительные вычислительные устройства – суперкомпьютеры реализованы на MPP архитектуре.

NUMA архитектура

NUMA (англ. *nonuniform memory access*) – система с неоднородным доступом к памяти это гибридная архитектура, которая совмещает достоинства систем с общей памятью (SMP) и относительную дешевизну систем с раздельной памятью (MPP).

Суть этой архитектуры – в особой организации памяти: память физически распределена по различным частям системы, но логически она является общей, так что пользователь видит единое адресное пространство.

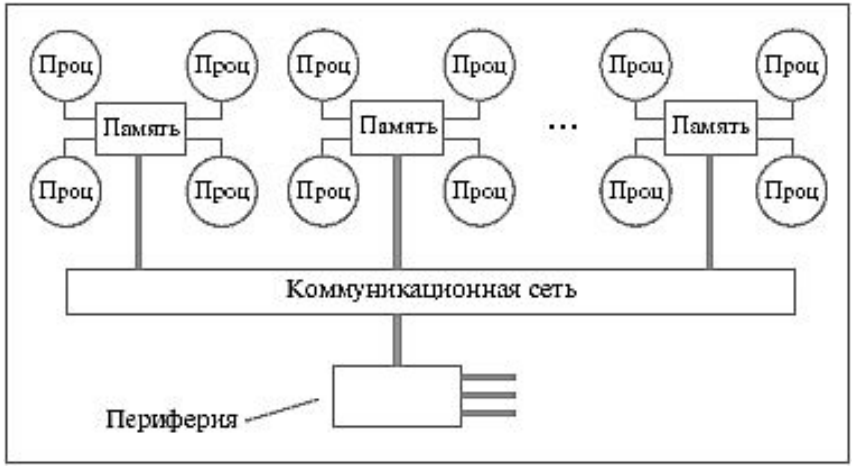


Рисунок 13 – NUMA архитектура ЭВМ

Система построена из однородных базовых модулей, состоящих из небольшого числа процессоров и единого блока памяти.

Модули объединены с помощью высокоскоростного коммутатора. Поддерживается единое адресное пространство, аппаратно поддерживается доступ к удаленной памяти, то есть к памяти других модулей. При этом доступ к локальной памяти осуществляется в несколько раз быстрее, чем к удаленной.

Доступ к памяти и обмен данными внутри одного SMP-узла осуществляется через локальную память узла и происходит очень быстро, а к процессорам другого SMP-узла тоже есть доступ, но более медленный и через более сложную систему адресации.

Распределённые вычисления

Метакомпьютинг – способ решения трудоёмких вычислительных задач с использованием нескольких компьютеров с различной архитектурой и различной мощностью, объединённых в параллельную вычислительную систему – сеть (англ. *grid*) для одновременного решения различных частей одной вычислительной задачи несколькими процессорами одного или нескольких компьютеров.

Метакомпьютер может не иметь постоянной конфигурации – отдельные компоненты могут включаться в его конфигурацию или отключаться от нее; при этом технологии метакомпьютинга обеспечивают непрерывное функционирование системы в целом.

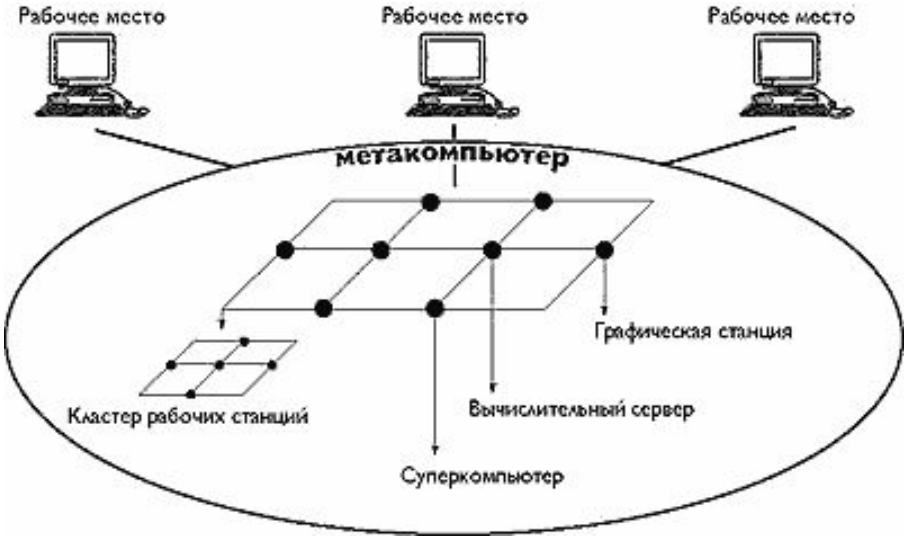
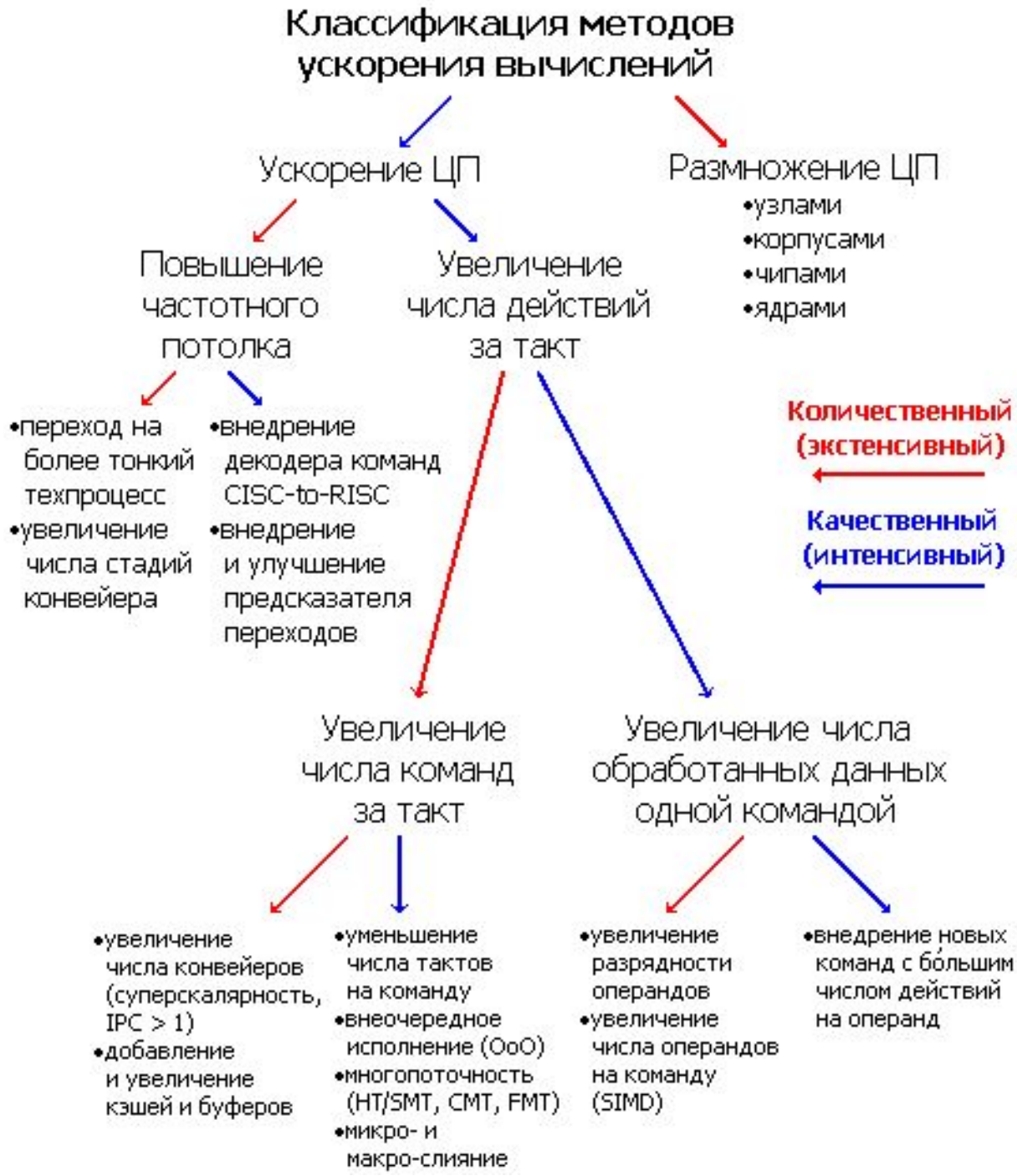


Рисунок 14 – Типы узлов метакомпьютера


Классификация методов ускорения вычислений
Ускорение ЦП
Размножение ЦП
•узлами
•корпусами
•чипами
•ядрами
Повышение частотного потолка
Увеличение числа действий за такт
•переход на более тонкий техпроцесс
•увеличение числа стадий конвейера
•внедрение декодера команд CISC-to-RISC
•внедрение и улучшение предсказателя переходов
Количественный (экстенсивный)
Качественный (интенсивный)
Увеличение числа команд за такт
Увеличение числа обработанных данных одной командой
•увеличение числа конвейеров (суперскалярность, IPC > 1)
•добавление и увеличение кэшей и буферов
•уменьшение числа тактов на команду
•внеочередное исполнение (OoO)
•многопоточность (HT/SMT, CMT, FMT)
•микро- и макро-слияние
•увеличение разрядности операндов
•увеличение числа операндов на команду (SIMD)
•внедрение новых команд с бо́льшим числом действий на операнд