## 1. Как выяснить, находится ли адресуемый объект (операнд) в КЭШе? (Как понять, что объект в кэше?)

1. ЦП(Центра́льный проце́ссор) обращается к объекту

2. Из адреса объекта выделяется три поля - **БАЙТ в блоке** памяти, **ИНДЕКС** и **ПРИЗНАК**.

Индекс служит для того, чтобы определить в какой кэш-линии расположен блок памяти (или, в общем случае, группа блоков памяти)

Признак служит для того, чтобы эти блоки памяти отличать между собой.

3. Следовательно, для того, чтобы определить, находится ли нужный в данный момент центральному процессору объект в кэше, нужно **потребовать полное совпадение всех этих полей**. Если они **не совпали**, то необходимо **обращаться к памяти**, а если **совпали**, то **объект передается** из кэша в ЦП.

*// ДРУГИМИ СЛОВАМИ : Процессор, выполняя команду, запрашивает операнд по некоторому адресу в адресном пространстве. Кэш-контроллер проверяет, есть ли в кэш-памяти строка данных, соответствующая запрашиваемому адресу. В случае наличия такой строки ситуация называется кэш-попадание. Если строки в кэш-памяти нет, то происходит кэш-промах, и кэш-контроллер инициирует обращение к основной памяти ОП для переписи из нее нужной строки в кэш-память.*

**КОРОТКИЙ ОТВЕТ НА ВОПРОС**: требуется совпадение всех составляющих адреса объекта с соответствующими значениями в кэш-памяти

Информационная память кэша и оперативная память вычислительной системы разбиты на блоки одинаковой длины и каждый блок памяти отображается на одну из строк кэша. В большинстве систем размер блока - 32 или 64 байта. Адрес объекта в памяти определяет строку кэша, в которую помещается блок информации, содержащий адресуемый объект.

1. При обращении центрального процессора к адресуемому объекту из адреса объекта выделяются три поля: - номер байта в блоке, индекс и признак.

— **номер байта в блоке**, занимающий младшие разряды адреса объекта;

— **индекс, определяющий адрес блока в кэше,** в котором должен находиться адресуемый объект, и занимает средние разряды адреса объекта;

— **признак,** занимающий старшие разряды адреса и позволяющий отличить блоки оперативки; при записи блока памяти в строку КЭШа, определяемую индексом, в элемент теговой памяти строки помещается признак.

2. По индексу, находится строка кэша, в которой должен находиться нужный блок памяти.

3. Признак, выделенный из адреса объекта, сравнивается с признаком, хранящимся в элементе признаковой памяти найденной строки кэша. Если признаки совпали, то это означает что адресуемый объект в кэше - произошло кэш-попадание. . Если же нет, то запрашиваемая информация в кэше отсутствует произошел кэш-промах.т.е. сравниваются Признак объекта выделенного из адреса и признаки находящиеся в служебных полях нашего кэша.

Промах чтения в кэш-памяти вызывает загрузку строки из основной памяти. В случае промаха записи транзакция записи помешается в буфер и посылается в основную память при предоставлении шины.

При чтении блока из памяти в кэш кроме записи самого блока нужно записать признак этого блока памяти.

Если длина элемента данных превышает один байт, то возможны ситуации, когда этот элемент (частями) расположен в двух (или более) разных КЭШ-строках, тогда время на выборку такого элемента увеличится. Противодействовать этому можно, выравнивая операнды и команды по границам КЭШ-строк.

# 2. Чем определяется объем кэшированной оперативной памяти?

В общем случае, объем кэшируемой ОП определяется разрядностью признаковой памяти кэша

Нетрудно видеть, что при А = 1 множественный ассоциативный кэш превращается в кэш прямого отображения, а при

А, равном числу строк в кэше, в полностью ассоциативный

кэш.

Количество групп в кэше является величиной противоположной коэффициенту А: полностью ассоциативный кэш состоит из одной группы, а кэш прямого отображения содержит

столько групп, сколько блоков в кэше. Соответственно с ростом

ассоциативности кэша размерность признакового поля растет, а

разрядность индекса падает (до О в полностью ассоциативном

кэше). С ростом длины блока при постоянной емкости кэша Kоличество строк кэша уменьшается, и разрядность индекса также

снижается, размер поля "номер байта в блоке" растет, а объем

признакового поля не изменяется.

# 3. Отображение памяти на кэш

## КЭШ прямого отображения.

ОСНОВНАЯ ИДЕЯ прямого отображения (direct mapping) RAM на кэш-память - RAM делится на сегменты, причем размер каждого сегмента равен размеру кэша, а каждый сегмент в свою очередь делится на блоки, размер каждого блока равен размеру кэш-линии. Блоки RAM из разных сегментов, но с одинаковыми номерами в этих сегментах, всегда будут отображаться на одну и ту же кэш-линию кэша. Адрес каждого байта представляет собой сумму порядкового номера сегмента, порядкового номера кэш-линии внутри сегмента и порядкового номера байта внутри кэш-линии. Отсюда следует, что адреса байт различаются только старшими частями, представляющими собой порядковые номера сегментов, а порядковые номера кэш-линий внутри сегментов и порядковые номера байт внутри кэш-линий — повторяются.

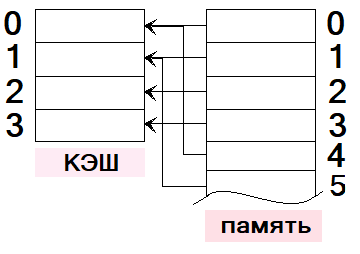
Таким образом нет необходимости хранить полный адрес кэш-линии, достаточно сохранить только старшую часть адреса.

**Каждой строке КЭШ памяти соответствуют конкретные, заранее определенные строки оперативной памяти**

ПЛЮСЫ : Требуется только **ОДНО СРАВНЕНИЕ** признаковых полей

МИНУСЫ : **КОНФЛИКТ АДРЕСОВ -**

Конкуренция **множества** блоков **ПАМЯТИ** на **одну** строку **КЭША**



//Как правило, не используется. **АЛЬТЕРНАТИВА - ПАК**.

### Полностью ассоциативный КЭШ**.**

ОСНОВНАЯ ИДЕЯ полностью ассоциативного отображения (fully associative mapping) RAM на кэш-память - RAM делится на блоки, размер которых равен размеру кэш-линий, а каждый блок RAM может сохраняться в любой кэш-линии кэша.

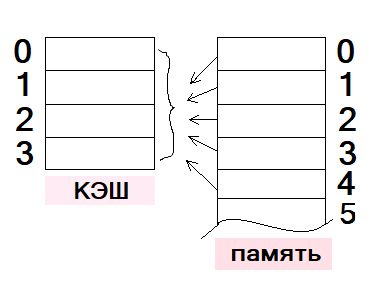
Адрес каждого байта представляет собой сумму порядкового номера кэш-линии и порядкового номера байта внутри кэш-линии. Отсюда следует, что адреса байт различаются только старшими частями, представляющими собой порядковые номера кэш-линий. Порядковые номера байт внутри кэш-линий повторяются.

**Каждая строка оперативной памяти может попадать на любую строку КЭШ**

ПЛЮСЫ : Строки **КЭША** и **ПАМЯТИ** НИКАК НЕ СВЯЗАНЫ

НЕ ВОЗНИКАЕТ КОНФЛИКТ АДРЕСОВ

МИНУСЫ : Признаковые поля **сравниваются ОЧЕНЬ МНОГО** раз

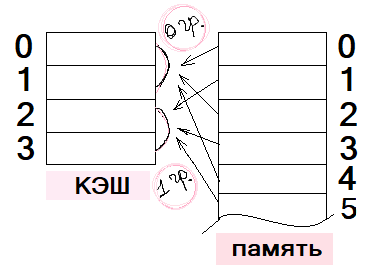


### Множественно ассоциативный КЭШ**.**

Множество строк кэша разбивается на A >= 1 строк.

Между группами - **КПО** ( кэш прямого отображения )

Внутри групп - **ПАК** ( полностью ассоциативный КЭШ )



ПЛЮСЫ : **КОНФЛИКТ** адресов **НИЖЕ**, чем у КПО

**СРАВНЕНИЙ МЕНЬШЕ**, чем у ПАК

Множество строк кэша разбивается на A >= 1 строк. А целочисленное – коэффициент ассоциативности кэша. А = 2 – двухвходовый множественный ассоциативный кэш, А = 4 – четырёхвходовый и т д. Между группами такой кэш представляет собой кэш прямого отображения, а внутри групп - кэш полностью ассоциативного отображения

Отличие множественного ассоциативного кэша от кэша прямого отображения в том, что индекс адресует номер группы, и адресуемый блок может располагаться в любой строке группы. Т.е. между группами множественный ассоциативный кэш является кэшем прямого отображения, а внутри группы — полностью ассоциативным. Из-за этого растет коэффициент удачных обращений к кэшу. Кроме того, количество сравнений старших разрядов адреса основной памяти с признаком из теговой памяти, в группах строк кэша, сокращается при ”промахе“, до размера группы. Оба этих фактора положительно влияют на производительность вычислительной системы. С ростом ассоциативности, растет частота ”попаданий“ в кэш, и расходы в виде количества сравнений признаков. Для снижения расходов при поиске нужной строки в каждой группе, аппаратно реализуется параллельное сравнение признаков, от этого дорожает кэш-память и растет ассоциативность.

# 4. В какой архитектуре КЭШа самый большой процент попадания?

Самый большой процент попадания в **МНОЖЕСТВЕННО-АССОЦИАТИВНЫЙ** КЭШ, так как сокращается количество коллизий блоков памяти, а поиск нужной кэш-линии внутри группы ограничивается ее размерами (и не обязательно составляет размер кэша, как это было в полностью ассоциативном кэш).

# 5. Как изменяется разновидность признака, индекса, блока с изменением емкости КЭШа, длины блока?

С ростом длины блока уменьшается признак, индекс и номер блока.

С ростом ассоциативности все поля увеличиваются.

С ростом емкости кэша - уменьшаются.

С **РОСТОМ АССОЦИАТИВНОСТИ** кэша размерность признакового поля растет, а разрядность индекса падает (до О в полностью ассоциативном кэше).

С **РОСТОМ ДЛИНЫ БЛОКА** при постоянной емкости кэша количество строк и разрядность индекса кэша уменьшается, размер поля "номер байта в блоке" растет, а объем признакового поля не изменяется.

# 6. Разница между архитектурами CISC / RISC (Соотношение CISC и RISC архитектур)

**CISC** – ЦП (центральный процессор) с **ПОЛНЫМ** набором команд.

Форматы команд : **ПАМЯТЬ-ПАМЯТЬ, ПАМЯТЬ-РЕГИСТР,**

**РЕГИСТР-РЕГИСТР**

Архитектура процессора, которая характеризуется следующим набором свойств:

1. **Короткая программа** - длинные команды (все команды разных форматов).

*добавление в арифметический конвейер*

***длинной цепочки*** *доступа к оперативной памяти неминуемо приведет к* ***низкой эффективности*** *ero функционирования.*

1. В одной команде - несколько арифметических действий.
2. Обращение к памяти разрешено для инструкций разного рода.
3. **Небольшое число регистров**, каждый из которых выполняет строго определённую функцию.

**RISC** – ЦП (центральный процессор) с **СОКРАЩЕННЫМ** набором команд

Один формат : **РЕГИСТР-РЕГИСТР**

Архитектура процессора, в которой **быстродействие увеличивается** за счёт УПРОЩЕНИЯ ИНСТРУКЦИЙ, чтобы их декодирование было более простым, а **время** выполнения — **меньшим**.

1. Длинная программа - **короткие команды (**фиксированный 32-битный формат у всех команд)
2. **ПОЛНОРАЗРЯДНАЯ ОБРАБОТКА** ДАННЫХ -

Каждый арифметический конвейер за ОДИН ТАКТ формирует ОДИН 64-разрядных результат (один такт = одно слово).

1. Обращение к памяти происходит посредством команд загрузки и записи
2. **Большое количество регистров** на уровне архитектуры.
3. **Конвейерная организация** арифметических и других исполнительных устройств.

**CISC** - архитектура вычислений с полной системой команд. Выполнение команд обработки данных, сохраняемых в памяти (операции "память­-память").

**RISC** - архитектура вычислений с сокращенным набором команд. Подразумевает обработку данных, хранящихся только в регистрах. **Обе архитектуры являются воплощением SISD-архитектур.** RISC- архитектура производительнее в тех приложениях, где над каждой единицей данных выполняется большой объем вычислительной работы (инженерные и научные задачи), в противном случае часто приходится загружать данные из памяти.

# 7. Способы увеличения производительности процессов

1. (?) КОНВЕЙЕРИЗАЦИЯ - это параллельное выполнение всех ступеней обработки в пределах одного и того же процесса.
2. СУПЕРСКАЛЯРНОСТЬ.

Суперскалярные процессоры - процессоры с несколькими конвейерами.

**Основная идея**, определяющая развитие суперскалярных микропроцессоров - построение **большего количества параллельных структур**, сохранение традиционных последовательных программ.

1. НЕУПОРЯДОЧЕННОЕ выполнение.

Архитектуры с **неупорядоченным завершением и исполнением** позволяют одному конвейеру продолжать работать при “заторе” в другом.

При этом команды, **стоящие позже**, могут быть **выполнены раньше** предыдущих - ЗАСТРЯВШИХ в другом конвейере.

// “Затор” - зависание, обусловленное объективными взаимозависимостями команд в программе и способами построения суперскалярных архитектур.

**!!!!!! ВАЖНО,** чтобы процессор гарантировал, что результаты запишутся в память, а регистры модифицируются в правильной последовательности!!!!!

1. ПЕРЕИМЕНОВАНИЕ регистров.

!! Частое **использование ОДНИХ регистров** приводит к возникновению ВЗАИМОЗАВИСИМОСТЕЙ и **СНИЖЕНИЮ ПРОИЗВОДИТЕЛЬНОСТИ** !!

**CISC-**архитектуры имеют **небольшое число** архитектурных **регистров** → ЛОЖНЫЕ взаимозависимости возникают довольно **часто**.

Путь **повышения производительности** - нейтрализация возникновения ЛОЖНЫХ взаимозависимостей путем ПЕРЕИМЕНОВАНИЯ РЕГИСТРОВ:

* **КОМАНДЕ 1** требуется **регистр** → процессор динамически **ставит в соответствие** этому АРХИТЕКТУРНОМУ (логическому) регистру один из множества ФИЗИЧЕСКИХ регистров.
* **КОМАНДА 2** пытается обратиться к **этому же** регистру → процессор **ставит в соответствие** другой ФИЗИЧЕСКИЙ регистр.

Такие переименования действуют, пока **команды продвигаются по конвейерам**

1. ОБХОДЫ и ПРОДВИЖЕНИЯ данных.

* **ОБХОД ДАННЫ**Х :

результат работы **КОМАНДЫ 1** сразу пересылается **КОМАНДЕ 2**.

Следовательно, **исключаются** ЗАДЕРЖКИ на **МОДИФИКАЦИЮ** и повторное **чтение** из регистра или памяти.

* **ПРОДВИЖЕНИЕ ДАННЫХ :**

процессор может **выполнять** некоторые команды **ПАРАЛЛЕЛЬНО**, немедленно передавая результаты работы команды 1 команде 2 (которой они потребуются на более поздней стадии конвейера)

Следовательно, **исключается** ожидание завершения операций ЗАПИСИ и ЧТЕНИЯ используемых операндов.

ОБХОДЯТСЯ ИСТИННЫЕ ЗАВИСИМОСТИ

1. ПРОГНОЗИРОВАНИЕ переходов.

Поскольку ВЕТВЛЕНИЯ (условные переходы) **разрешаются на исполнительной** ступени конвейера, **процессор не знает**, какие команды отправлять в конвейер за командой перехода.

Для этого и **используется ПРОГНОЗИРОВАНИЕ ПЕРЕХОДОВ** (предположение о пути ветвления), при котором процессор начинает :

* **ЧТЕНИЕ команд** с предсказанного адреса до того, как узнает, верным ли был прогноз.
* **ИСПОЛНЕНИЕ** по предположению “спекулятивное”.

//При этом процессор **не может модифицировать** архитектурные //**РЕГИСТРЫ** или **ПАМЯТЬ** до уточнения достоверности предсказания.

1. (?) ПРЕВРАЩЕНИЕ **CISC → в RISC**

**ГЛАВНАЯ ПРОБЛЕМА** декодирования **CISC**-команд - **определить** границы команд и **расположить** их в последовательности, которая обеспечивает эффективную ОБРАБОТКУ **ДЕШИФРАТОРОМ** (быстрое декодирование - **обязательное условие** обработки нескольких команд за такт)

**Предварительное декодирование** - распознавание сложных команд до стадии декодирования на этапе загрузки из основной памяти в кэш-память. Процесс **НЕ СНИЖАЕТ производительность** системы, но **требует** дополнительной **КЭШ**-памяти.

# 8. Стадии, фазы конвейера

**КОНВЕЙЕРИЗАЦИЯ**- это параллельное выполнение всех ступеней обработки в пределах одного и того же процесса.

Каждая СТУПЕНЬ обрабатывает свою ФАЗУ **независимо**.

В каждый момент времени конвейерный процессор работает над выполнением различных СТАДИЙ **нескольких команд**.

**КОНВЕЙЕР** - команда из 5 классических **СТАДИЙ**:

1. **ЧТЕНИЕ** КОМАНД
2. **ДЕШИФРОВАНИЕ**
3. **ЧТЕНИЕ** ОПЕРАНДОВ
4. **ИСПОЛНЕНИЕ**
5. **ЗАПИСЬ РЕЗУЛЬТАТА**

Различные **КОМАНДЫ** - неодинаковое **ВРЕМЯ** выполнения различных фаз.

Максимальная ЭФФЕКТИВНОСТЬ в случае, когда **время** выполнение ФАЗ **ОДИНАКОВО (однородный** конвейер).

К факторам, нарушающим непрерывность конвейерной реализации команд, относятся случаи, когда :

* Для выполнения команды 2 требуется завершение команды 1.
* В кэш-памяти отсутствуют некоторые данные, и требуется передать их из ОСНОВНОЙ памяти в КЭШ-память.
* При ВЕТВЛЕНИИ, по результатам проверки условий, команды, находящиеся в процессе конвейерной обработки, остаются не выполненными. Требуется снова загружать конвейер с момента чтения команды условного перехода.

## Фазы классического конвейера

В большинстве классических СISC-архитектур имеется пять ступеней обработки конвейера:

1) Выборка команды

2) Декодирование команды

3) Адресация и выборка операндов

4) Исполнение команды

5) Запись результата

# 9. Истинные и ложные зависимости

Причины **ограничения производительности** процессора :

1. **ИСТИННАЯ** взаимозависимость данных -

ВХОДНЫЕ данные **для одной** операции зависят от РЕЗУЛЬТАТА **другой** операции

(**к примеру**, в выражении **n\*k+m** НЕЛЬЗЯ выполнить сложение и умножение одновременно)

“ЧТЕНИЕ ПОСЛЕ ЗАПИСИ”

1. **ЛОЖНАЯ** взаимозависимость данных:

а) ВЗАИМОЗАВИСИМОСТЬ **ПО ВЫХОДУ:**

Команды, следующие друг за другом записывают результат в **один и тот же** РЕГИСТР

“ЗАПИСЬ ПОСЛЕ ЗАПИСИ”

б) **АНТИ** ВЗАИМОЗАВИСИМОСТЬ:

**Команда 1** может **испортить** данные, необходимые для ВХОДА **команде 2.**

“ЗАПИСЬ ПОСЛЕ ЧТЕНИЯ”

## Ложные и истинные взаимозависимости Как борются с истинными взаимозависимостями?

**Истинная взаимозависимость данных** или взаимозависимость “чтение после записи” **-**входные данные для одной операции зависят от результата другой, (например – a\*b+c). **Анти-взаимозависимость**, или взаимозависимость “ запись после чтения“ возникает, когда вторая команда может испортить данные, необходимые в качестве входных для первой. **Переименование регистров** – нейтрализация ложных взаимозависимостей путем переименования регистров. RISC-процессоры имеют достаточное кол-во регистров чтобы ложные взаимозависимости не появлялись часто, в то время как CISC-процессоры имеют **небольшое кол-во архитектурных регистров, что провоцирует частое возникновение ложных взаимозависимостей**. Если команде требуется регистр, процессор динамически ставит в соответствие этому логическому регистру один из множества физических регистров. Если другая команда пытается обратиться к тому же логическому регистру, то процессор для предотвращения конфликта может поставить ему в соответствие другой физический регистр. Такие переименования действуют, пока команды продвигаются по конвейерам. Только переименование регистров не решает проблемы ложной взаимозависимости, поэтому используется (ист) **обход данных –** результаты выполнения одной команды сразу пересылаются следующей и таким образом исключаются задержки на модификацию и повторное чтение из регистра или памяти. Так же используется (ист) **продвижение данных** - процессор может выполнять некоторые команды параллельно, немедленно передавая результаты одной из них в другую, которой они потребуются на более поздней ступени конвейерной обработки. При этом также исключается ожидание завершения операций записи и чтения адресуемых операндов.

# 10. Чем отличается модель программирования UMA от NUMA?

UMA - **Uniform Memory Access - РАВНОМЕРНЫЙ** доступ к памяти

Архитектура **С** **ОБЩЕЙ ПАМЯТЬЮ** (использует один контроллер памяти), к которой обращаются **ВСЕ ПРОЦЕССОРЫ** представленной многопроцессорной системы с помощью межсоединительной сети.

Каждый процессор имеет **РАВНОЕ** время доступа к памяти (задержка) и скорость доступа. Он может использовать одну шину / несколько шин / коммутатор.

**Пропускная способность** ОГРАНИЧЕНА.

NUMA - **Non** **Uniform Memory Access - НЕРАВНОМЕРНЫЙ** доступ к памяти

Многопроцессорная модель, в которой **каждый ПРОЦЕССОР** связан с **СОБСТВЕННОЙ** (выделенной) памятью (контроллер множественной памяти).

Все части памяти объединяются в единое АДРЕСНОЕ ПРОСТРАНСТВО.

**В отличие от UMA,** время доступа к памяти **зависит от расстояния**, на котором расположен процессор, что означает изменение времени доступа к памяти. Это позволяет получить доступ к любой ячейке памяти, используя физический адрес.

**Пропускная способность** БОЛЬШЕ, ЧЕМ У UMA.

**СС-NUМА** -системы с кэш когерентным доступом (при изменении объекта одним процессором или ядром, изменяется объект для всей системы) к неоднородной памяти. Системы этого класса строятся из узлов с SМР­ архитектурой, имеющих собственную (локальную) область общего адресного пространства и кэш "дальней" памяти (областей памяти других узлов). Узлы между собой связываются с помощью шины. По существу, данная архитектура является расширением систем с симметричной мультипроцессорностью. В отличие от классической архитектуры NUMA, при использовании кэш-когерентного доступа к неоднородной памяти все процессоры объединены в один узел, причем первый уровень иерархии памяти образует кэш-память процессоров.

**UMA** - системы с однородной памятью. Имеют общую (разделяемую) память для всех процессоров вычислителя. Архитектура предполагает наличие одной (реже двух или более) общей шины к которой подключены несколько процессоров. Доступ к любому элементу данных происходит единообразно.

**NUMA** –системас неоднородным доступом к памя­ти. Наличие индивидуальной оперативной памяти возле каждого процессора (или группы процессоров) и поддержка неоднородного доступа к памяти, обеспечивающая существенно различное время обращения к объектам в распределенной памяти. Каждый процессор имеет доступ не только к собственной ло­кальной памяти, но и к памяти других процессоров сети. Но поскольку при обращении к памяти других процессоров запросы проходят через сеть, они вы­полняются дольше, чем обращения к локальной памяти. Системы этого типа называются мультипроцессорными системами. В системе CC-NUMA **физически распределенная память объединяется в единый массив** (единое адресное пространство с частями, физически связанными специальной соединительной шиной, и аппаратно-реализованная кэш-когерентность).

**С точки зрения модели программирования:**

-Ключевое различие – диктуемая модель программирования. Задержки доступа к адресуемым объектам происходят из-за различия в способах программирования.

· SMP-, СМР-и СС-МЈМА-системы попадают в один класс компьютерных систем с общей памятью и аппаратно-поддерживаемой когерентностью данных. (+ доступ к часто используемым данным, в разделяемой памяти, за микросекунды, в то время как считывание их с диска требует миллисекунд)

· МРР-системы и вычислительные кластеры — в класс вычислительных систем с распределенной памятью и программно-реализуемой когерентностью данных (-время доступа к объектам в удаленной памяти повышается до сотни микросекунд — это в сотни раз медленнее, чем быстродействие локальной памяти)

**Наиболее выгодно программирование в SMP**, поскольку при этом **программисту не нужно заботиться о распределении данных в памяти**, так как все ее части доступны для любого процессора и доступ к ним одинаково быстр.

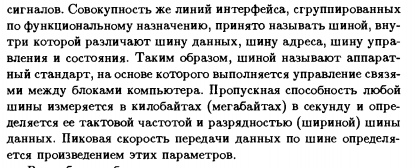
# 11. Чем определяется быстродействие истинного интерфейса?

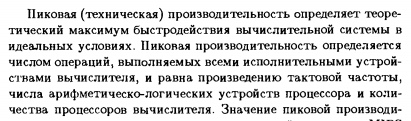
ВАЖНЕЙШИМ фактором, определяющим БЫСТРОДЕЙСТВИЕ **ВС**, является ЁМКОСТЬ КЭША (?)

ВАЖНЕЙШИМ фактором, определяющим БЫСТРОДЕЙСТВИЕ

**дискового накопителя**, является ПЛОТНОСТЬ ЗАПИСИ.

// Если **ИСТИННЫЙ** интерфейс = интерфейс самого **НИЗКОГО уровня**, тогда ответ “кэш первого уровня”





# 12. Для чего нужен обход и продвижение данных в ЦП?

Переименование регистров **не полностью исключает** возникновение ИСТИННЫХ взаимозависимостей.

Поэтому используются методы ОБХОДА и ПРОДВИЖЕНИЯ данных.

* **ОБХОД ДАННЫ**Х :

результат работы **КОМАНДЫ 1** сразу пересылается **КОМАНДЕ 2**.

Следовательно, **исключаются** ЗАДЕРЖКИ на **МОДИФИКАЦИЮ** и повторное **чтение**.

* **ПРОДВИЖЕНИЕ ДАННЫХ :**

процессор может **выполнять** некоторые команды **ПАРАЛЛЕЛЬНО**, немедленно передавая результаты работы команд.

Следовательно, **исключается** ожидание завершения операций ЗАПИСИ и ЧТЕНИЯ используемых операндов.

# 13. В каком случае используется прогнозирование переходов и для чего это нужно?

Приемы **увеличения производительности**\* **имеют смысл,** только когда решена проблема процедурных взаимозависимостей, возникающая из-за **наличия** условных и безусловных **ПЕРЕХОДОВ** в программе.

Поскольку ВЕТВЛЕНИЯ (условные переходы) **разрешаются на исполнительной** ступени конвейера, **процессор не знает**, какие команды отправлять в конвейер за командой перехода.

Для этого и **используется ПРОГНОЗИРОВАНИЕ ПЕРЕХОДОВ** (предположение о пути ветвления), при котором процессор начинает :

* **ЧТЕНИЕ команд** с предсказанного адреса до того, как узнает, верным ли был прогноз.
* **ИСПОЛНЕНИЕ** по предположению “спекулятивное”.

//При этом процессор **не может модифицировать** архитектурные //**РЕГИСТРЫ** или **ПАМЯТЬ** до уточнения достоверности предсказания.

**ПРИЕМЫ** прогнозирования ветвлений:

* Использование буфера адреса перехода
* Буфер адресов переходов дополняется стеком возвратов, который отслеживает переходы в паре команд CALL/RETURN
* Информация о переходах добавляется к каждой строке КЭШа команд

МЕТОДЫ:

**Статические методы** предписывают всегда выполнять или не выполнять определенные типы переходов.

**Динамические методы** оценивают поведение команд перехода за предшествующий период времени.

\*СУПЕРСКАЛЯРНОСТЬ, НЕУПОРЯДОЧЕННАЯ ОБРАБОТКА, ПРОДВИЖЕНИЕ данных

## Зачем нужно прогнозирование?

Предсказание ветвлений позволяет сократить время простоя конвейера за счёт предварительной загрузки и исполнения инструкций, которые должны выполниться после выполнения инструкции условного перехода. Без предсказания переходов конвейер должен дождаться выполнения инструкции условного перехода, чтобы произвести следующую выборку. Предсказатель переходов позволяет избежать траты времени, пытаясь выяснить ответвление. Ответвление выбирается по предыдущим результатам проверки условия. Предполагаемое ответвление затем загружается и частично выполняется. Если затем обнаруживается, что предсказание было выполнено неверно, отменяются результаты неверного ветвления и в конвейер загружается правильное ответвление, производя задержку. Существует 2 метода прогнозирования:

* **Статические методы** предписывают всегда выполнять или не выполнять определенные типы переходов.
* **Динамические методы** оценивают поведение команд перехода за предшествующий период времени.

В Р6 используется исполнение по предположению — исполнение команд, следующих за командой условного перехода до того, как выяснится, правильно ли был предсказан переход. При этом процессор не обновляет архитектурные регистры и память до однозначного разрешения команд, выполненных по предположению. При неправильном предсказании ROB процессора Р6 отбрасывает исполненные по предположению команды, прежде чем они будут удалены.

Как происходит:

Устраняет процедурные зависимости, то есть условные и безусловные переходы. Увеличивает эффективность суперскалярности, неупорядоченной обработки и продвижения данных. Процессор использует “предложение о пути ветвления” и после этого начинает: 1. Выборку команд с предсказанного адреса перехода до того, как он узнает, верным ли был его прогноз. 2. Исполнение по предположению “спекулятивное”, при этом процессор не может модифицировать архитектурные регистры или память до однозначного разрешения перехода. Обычно используется буфер адреса перехода. Бывает предикатное выполнение команд: выполнение по всем возможным исходам условного перехода.

# ШИННЫЙ ИНТЕРФЕЙС

**Системная шина**Объединяющая центральный процессор с памятью и интерфейсом подключения внешних устройств;

**ШИНЫ ВВОДА/ВЫВОДА**

В современных вычислительных системах шину ввода-вывода называют также шиной РАСШИРЕНИЯ, поскольку она позволяет подключить к компьютеру широкую номенклатуру дополнительных периферийных устройств.

**ПЕРИФЕРИЙНЫЕ ШИНЫ**

Периферийные шины являются компонентами следующего уровня и располагаются между интерфейсными платами контроллеров и периферийными устройствами. Эти интерфейсы изменяются так же быстро, как и остальная часть вычислительной системы

**Различают шину данных, шину адреса, шину управления и состояния. Шиной называют аппаратный стандарт, на основе которого выполняется управление связями между блоками компьютера**

**3.2. Шины ввода-вывода**

Она же шина расширения используется в качестве шины для подключения контроллеров внешней периферии.

**3.2.1. Шина PCI и унаследованные шины**

Шина РС1 стала стандартом подключения внешних устройств для вычислительных архитектур. Для настольных систем реализуется обычно 32-разрядная версия стандарта, работающая на частоте 33 МГц, а для серверных платформ и высокопроизводительных рабочих станций используется четыре версии: 32 бит/ЗЗ МГц, 32 бит/бб МГц, 64 бит/ЗЗ МГц и самая скоростная 64 бит/бб МГц.

Спецификация этой шины (PCI-X) увеличивает производительность (до 1 Гбайт/с. Два главных отличия PCI-X от PCI состоят в использовании межрегистрового протокола (декодирование производится за отдельный цикл, тогда как раньше оно производилось на протяжении того же цикла) и атрибутивной фазы(во время этой фазы поле длиной 36 бит сообщает более подробную информацию о транзакции , чем это предусмотрено прежней спецификацией IPC).

**3.2.2. Ускоренный графический порт AGP**

Шина обмена графической информацией. Порт AGP имеет самостоятельный доступ к памяти. Разрядность шины составляет 32 бит, рабочая частота — 66 МГц, в шине поддерживаются режимы передачи х1, х2, х4. Порт AGP предназначен для обмена данными между основной памятью и видеоадаптером.

**3.3.1. Шина EIDE**

Используется для подключения устройств долговременной памяти является расширенный интерфейс подключения к компьютерам АТ накопителей со встроенным контроллером. Данная шина имеет два канала, к каждому из которых можно подключить по два устройства.

Теоретическая пропускная способность в настоящее время составляет 16,7 Мбайт/с. Длина интерфейсного кабеля до 45 см.

Недостаток: значительную часть работы по управлению выполняет процессор.

**Из чего состоит параллельная шина?**

По способу передачи данных шины делятся на последовательные и параллельные. Последовательные шины передают данные по одному проводнику, один бит за один раз, в параллельных шинах передача данных разделена между несколькими проводниками и поэтому можно передать большее количество данных.

# 1. Классификация ВС по Флинну.

Базируется на понятии **ПОТОКА**, под которым понимается **последовательность** элементов, команд или данных, обрабатываемая процессором.



SISD

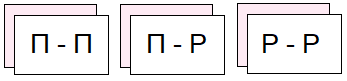
**В настоящее время имеется три воплощения SISD­ - архитектур:**

1. **CISC** - ЦП с **ПОЛНЫМ** набором команд.

Концепция проектирования процессоров, которая характеризуется следующим набором свойств:

1. Короткая программа - длинные команды.
2. Арифметические действия кодируются в одной команде.
3. **Небольшое число регистров**, каждый из которых выполняет строго определённую функцию.

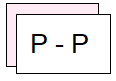
Формат команд (п- память, р - регистр) :



1. **RISC** - ЦП с СОКРАЩЕННЫМ набором команд.

Архитектура процессора, в которой быстродействие увеличивается за счёт упрощения инструкций, чтобы их декодирование было более простым, а время выполнения — меньшим.

Один формат : р - регистр



* ЗАГРУЗКА и ВЫГРУЗКА

1. **СУПЕРСКАЛЯРНЫЕ** архитектуры - больше одного КОНВЕЙЕРА
2. с **ДИНАМИЧЕСКИМ** предсказанием ветвлений
3. со **СТАТИЧЕСКИМ** предсказанием ветвлений.

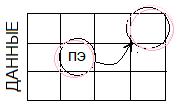
Основан на VLIW-архитектуре.

1. **VLIW -** архитектура с очень длинным (широким) командным словом **до n команд.**

Реализуется КОМПИЛЯТОРОМ.

#### SIMD. ВАРИАНТЫ РЕАЛИЗАЦИИ :

* **МАТРИЧНАЯ** архитектура команды



**ПЭ** - процессорные элементы

ВЫСОКАЯ **сложность** коммутировать результаты вычислений между ПЭ

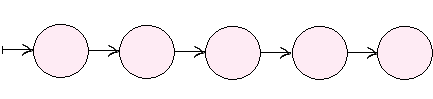
* **ВЕКТОРНО-КОНВЕЙЕРНАЯ** архитектура

ВЕКТОРИЗАЦИЯ матричных вычислений

**ОСНОВА -** длинные регистры **(множество швов)**

КОНВЕЙЕРИЗАЦИЯ :

Запускаем одну и ту же программу разными словами



#### MISD -

специальные решения, обеспечивающие **ГОТОВНОСТЬ** и **НАДЕЖНОСТЬ** ВС.

Самая простая и **распространенная система** этого класса - ЛОКАЛЬНАЯ ВЫЧИСЛИТЕЛЬНАЯ СЕТЬ персональных компьютеров, работающая **с единой базой** данных.

#### MIMD-

включает **ВСЕ УРОВНИ** параллелизма (**от конвейера** операций **до независимых** заданий и **программ**)

ВС, построенные на основе MIMD-архитектуры :

1. ВС с **СИЛЬНОЙ СВЯЗЬЮ** процессора с оперативной памятью ВС.

SMP-вычислители - **ОБЩАЯ** разделяемая память, имеющая единое адресное пространство.

При **увеличении числа процессов** становится **узким местом системы** процессов не больше нескольких десятков.

1. ВС со **СЛАБОЙ СВЯЗЬЮ** процессора с оперативной памятью ВС

МРР-вычислители -ВС с **распределенной** памятью **по узлам.**

**В каждом узле:**

время доступа к ЛОКАЛЬНОЙ памяти = **t**

время доступа к УДАЛЕННОЙ памяти = **Т**

// где Т>>t

**УЗЛЫ** объединяются между собой **КОММУТАЦИОННЫМИ** сетями регулярной структуры :

* ДЕРЕВО
* КОЛЬЦО
* ГИПЕРКУБ
* РЕШЕТКА
* ТОР

# 2. Особенности построения CISC архитектуры.

**CISC-архитектура** – это концепция проектирования процессоров, которая характеризуется следующим набором свойств:

1. **Короткая программа** - длинные команды.
2. **Пе**ренос вычислений ПРОГРАММНОГО уровня на АППАРАТНЫЙ.

**Результат**  форматы команд П-П, Р-Р, П-Р.

1. **ПРИНЦИП “80/20”:**

**“80%** кода команд используют **только 20%** ПРОСТЫХ команд **РЕГИСТР-РЕГИСТР** полного набора CISC-инструкций.”

1. Арифметические действия кодируются в **одной** команде.
2. **Небольшое число регистров**, каждый из которых выполняет строго определённую функцию.

**Любой CISC-процессор** УСТУПАЕТ **RISC-процессорам** по количеству выполняемых операций в секунду.

# 3. Особенности построения RISC-архитектуры.

**RISC-архитектура -**

Архитектура процессора, в которой **быстродействие увеличивается** за счёт УПРОЩЕНИЯ ИНСТРУКЦИЙ, чтобы их декодирование было более простым, а **время** выполнения — **меньшим**.

**ПРИНЦИПЫ RISC-архитектуры:**

1. Длинная программа - **короткие команды**
2. Любая операция **выполняется за ОДИН** такт.
3. Система команд должна содержать минимальный набор **ПРОСТЕЙШИХ КОМАНД** (инструкций) с одинаковой длиной и минимумом адресных форматов.
4. **ОБРАБОТКА** данных **только** в формате **“РЕГИСТР-РЕГИСТР”.**

Обмен “РЕГИСТР-ПАМЯТЬ” выполняется только с помощью команд “ЗАГРУЗКИ-ЗАПИСИ”

1. **Состав команд** должен быть **УДОБЕН** для компиляции операторов языков высокого уровня.

**Достижение возможного максимума производительности путем использования структурных приемов:**

ПОЛНОРАЗРЯДНАЯ ОБРАБОТКА ДАННЫХ -

Каждый арифметический **конвейер** за ОДИН **ТАКТ** формирует ОДИН **64-разрядных результат** (один такт = одно слово).

Конвейерная организация арифметических и других исполнительных устройств.

Суперскалярная структура процессора с

* комплексом средств динамического предсказания ветвлений
* большим числом регистров (min 32)
* многоуровневым КЭШем

**Набор команд** РАСШИРЯЕТСЯ за счёт **ПЕРЕРАСПРЕДЕЛЕНИЯ** площади кристалла.

# 4. КОНВЕЙЕРИЗАЦИЯ. ФАЗЫ КЛАССИЧЕСКОГО КОНВЕЙЕРА.

**КОНВЕЙЕРИЗАЦИЯ**- это параллельное выполнение всех ступеней обработки в пределах одного и того же процесса.

Каждая СТУПЕНЬ обрабатывает свою ФАЗУ **независимо**.

Различные КОМАНДЫ - различное ВРЕМЯ выполнения различных фаз.

Основные ступени обработки конвейера:

1. **ЧТЕНИЕ** КОМАНД
2. **ДЕШИФРОВАНИЕ**
3. **ЧТЕНИЕ** ОПЕРАНДОВ
4. **ИСПОЛНЕНИЕ**
5. **ЗАПИСЬ РЕЗУЛЬТАТА**

В большинстве классических СISC-архитектур имеется пять ступеней обработки конвейера:

1) Выборка команды

2) Декодирование команды

3) Адресация и выборка операндов

4) Исполнение команды

5) Запись результата

9. Методы предсказания ветвлений.  
 Существует два основных метода предсказания переходов — **статический** и **динамический**. **Статические методы** предписывают всегда выполнять или не выполнять определенные типы переходов. **Динамические методы** оценивают поведение команд перехода за предшествующий период времени

# 11. Переименование регистров.

**Переименование регистров** – нейтрализация ложных взаимозависимостей путем переименования регистров.

Причины **ограничения производительности** процессора :

1. **ИСТИННАЯ** взаимозависимость данных -

ВХОДНЫЕ данные **для одной** операции зависят от РЕЗУЛЬТАТА **другой** операции.

“ЧТЕНИЕ ПОСЛЕ ЗАПИСИ”

1. **ЛОЖНАЯ** взаимозависимость данных:

а) ВЗАИМОЗАВИСИМОСТЬ **ПО ВЫХОДУ:**

Команды, следующие друг за другом записывают результат в **один и тот же** РЕГИСТР

“ЗАПИСЬ ПОСЛЕ ЗАПИСИ”

б) **АНТИ** ВЗАИМОЗАВИСИМОСТЬ:

**Команда 1** может **испортить** данные, необходимые для ВХОДА **команде 2.**

“ЗАПИСЬ ПОСЛЕ ЧТЕНИЯ”

!! Частое **использование ОДНИХ регистров** приводит к возникновению ВЗАИМОЗАВИСИМОСТЕЙ и **СНИЖЕНИЮ ПРОИЗВОДИТЕЛЬНОСТИ** !!

**CISC-**архитектуры имеют **небольшое число** архитектурных **регистров** → ЛОЖНЫЕ взаимозависимости возникают довольно **часто**.

Путь **повышения производительности** - нейтрализация возникновения ЛОЖНЫХ взаимозависимостей путем ПЕРЕИМЕНОВАНИЯ РЕГИСТРОВ:

1. **КОМАНДЕ 1** требуется **регистр** → процессор динамически **ставит в соответствие** этому АРХИТЕКТУРНОМУ (логическому) регистру один из множества ФИЗИЧЕСКИХ регистров.
2. **КОМАНДА 2** пытается обратиться к **этому же** регистру → процессор **ставит в соответствие** другой ФИЗИЧЕСКИЙ регистр.

# 12. Обход и продвижение данных.

переименование регистров не решает проблемы ложной взаимозависимости, поэтому используется **обход данных –** результаты выполнения одной команды сразу пересылаются следующей и таким образом исключаются задержки на модификацию и повторное чтение из регистра или памяти. Так же используется **продвижение данных** - процессор может выполнять некоторые команды параллельно, немедленно передавая результаты одной из них в другую, которой они потребуются на более поздней ступени конвейерной обработки. При этом также исключается ожидание завершения операций записи и чтения адресуемых операндов.

Переименование регистров **не полностью исключает** возникновение ИСТИННЫХ взаимозависимостей.

Поэтому используются методы ОБХОДА и ПРОДВИЖЕНИЯ данных.

* **ОБХОД ДАННЫ**Х :

результат работы **КОМАНДЫ 1** сразу пересылается **КОМАНДЕ 2**.

Следовательно, **исключаются** ЗАДЕРЖКИ на **МОДИФИКАЦИЮ** и повторное **чтение**.

* **ПРОДВИЖЕНИЕ ДАННЫХ :**

процессор может **выполнять** некоторые команды **ПАРАЛЛЕЛЬНО**, немедленно передавая результаты работы команд.

Следовательно, **исключается** ожидание завершения операций ЗАПИСИ и ЧТЕНИЯ используемых операндов.

# 41. КЭШ память прямой и обратной записи.

Обратная запись дает инструкции процессору записать данные только в процессорный кэш, минуя основную оперативную память. Прямой способ записи (write-through) обеспечивает запись данных одновременно в кэш память процессора и в оперативную память системы. Первый способ отложенной записи избавляет от лишней нагрузки на оперативную память компьютера, хотя очень сложен в разработке и реализации

**Как загрузить объект в кэш? Чем отличается сквозная запись от обратной?**

С помощью методов сквозной и обратной записи.

1. **Метод сквозной записи** - – информация записывается в блок кэш-памяти и в оперативную память. Однако это снижает производительность системы, т.к. при каждой записи необходимо тратить время на запись в оперативную память.

2. **Метод обратной записи** - – информация записывается в блок кэш-памяти только при вытеснении его из кэша и при наличии в нем изменений. Для этого каждому блоку кэша ставится бит изменений, который устанавливается в случае записи новых данных. При замене старого блока информации кэша на новый из оперативки проверяется состояние соответствующею бита. Если он установлен, то сначала выполняется перепись блока из кэша в память, и только после этого в кэш помешается новый блок из памяти. Такой метод обеспечивает более высокую производительность вычислительной системы, так как количество измененных блоков обычно меньше числа операций записи в память.

# 42. Раздельная и совместная (с точки зрения инструкций и данных) КЭШ память.

Когда в микропроцессорах впервые стали применять внутреннюю кэш-память, ее обычно использовали как для команд, так и для данных; Такую кэш-память принято называть совместной. Сравнительно недавно стало обычным разделять кэш-память на две — отдельно для команд и отдельно для данных. Такая кэш-память получила название раздельной. Смешанная кэш-память обладает тем преимуществом, что при заданной емкости ей свойственна более высокая вероятность попаданий по сравнению с разделенной, поскольку в ней оптимальный баланс между командами и данными устанавливается автоматически. Так, если в выполняемом фрагменте программы обращения к памяти связаны в основном с выборкой команд, а доля обращений к данным относительно мала, кэш-память имеет тенденцию насыщаться командами, и наоборот.

С другой стороны, при раздельной кэш-памяти выборка команд и данных может производиться одновременно, при этом исключаются возможные конфликты. Последнее обстоятельство существенно в системах, использующих конвейеризацию команд, где процессор извлекает команды с опережением и заполняет ими буфер или конвейер.

# 43. Включающий и исключающий методы построения КЭШ памяти.

Пусть кэш имеет **включающую архитектуру**. Рассмотрим, каким образом происходит запись данных из оперативной памяти в такой кэш. Если в такой системе кэш ­памяти при полностью заполненном кэше L2 процессор пытается загрузить еще одну кэш ­строку, то произойдет следующее. Обнаружив, что все кэш­ строки заняты, кэш L2 избавляется от наименее ценной из них, стремясь при этом найти линейку, которая еще не была модифицирована, поскольку в противном случае ее еще придется выгружать в оперативную память.

Затем кэш L2 передает полученные из памяти данные кэшу L1. Если кэш первого уровня также заполнен, ему приходится избавляться от одной из кэш ­строк по сценарию, описанному выше.

Таким образом, **загруженная порция данных присутствует и в кэше L1, и в кэше L2.**

В случае кэша, построенного **по исключающей архитектуре**, кэш L1 никогда не уничтожает кэш ­строки при нехватке места. Даже если кэш­ строки не были модифицированы, они вытесняются в кэш L2 на то место, где находилась только что переданная кэшу L1 кэш­ строка. То есть **кэши L1 и L2 как бы обмениваются друг с другом своими кэш­ строками, благодаря чему кэш ­память используется весьма эффективно.**

# 44. Факторы, влияющие на выбор размера КЭШ памяти процессоров и ядер.

Оценивая размер кэш-памяти, нужно учитывать характеристики процессора и круг решаемых им задач.

Кэш-память двуядерного процессора редко превышает 3 MB. Тем более, если его тактовая частота ниже 3 Ггц. Производители прекрасно понимают, что дальнейшее увеличение размера кэша такого процессора не принесет прироста производительности, зато существенно повысит его стоимость.

Другое дело высокочастотные 4-, 6- или даже 8-миядерные процессоры. Некоторые из них (например, Intel Core i7) поддерживают технологию Hyper Threading, обеспечивающую одновременное выполнение каждым ядром двух задач. Естественно, что потенциал таких процессоров не может быть раскрыт с маленьким кэшем. Поэтому его увеличение до 15 или даже 20 MB вполне оправдано.

В целом же, эксперименты свидетельствуют, что в среднестатистическом "домашнем" процессоре влияние размера кэша на производительность находится в пределах 10 %, и его вполне можно компенсировать, например, высокой частотой.

Эффект от большого кэша наиболее ощутим при использовании архиваторов, в 3D играх, во время кодирования видео. В "не тяжелых" же приложениях разница стремится к нулю (офисные программы, интернет-серфинг, работа с фотографиями, прослушивание музыки и др.).

Многоядерные процессоры с большим кэшем необходимы на компьютерах, предназначенных для выполнения многопоточных приложений, одновременного решения нескольких сложных задач.

# 48. Механизмы обеспечения когерентности КЭШ памяти, протокол MESI.

В современных компьютерах доступ к памяти могут одновременно иметь несколько независимых процессоров (ядер, трэдов). Каждый из них имеет свои приватные кэши, в которых хранятся копии необходимых линий, а некоторые из них при этом локально модифицированы. Встает вопрос, а что если одна и та же линия одновременно понадобится нескольким процессорам. Не сложно сделать вывод, что для корректной работы системы необходимо обеспечить единое пространство памяти для всех процессоров.

Для обеспечения этого были придуманы специальные протоколы когерентности. **Когерентность кэша — свойства кэш-памяти, означающее целостность данных, хранящихся в локальных кэшах, разделяемой системы.** Каждая ячейка кэша имеет флаги, описывающие, как ее состояние соотносится с состоянием ячейки с таким же адресом в других процессорах системы.

При изменении состояния текущей ячейки необходимо каким-то образом сообщить об этом остальным кэшам. Например, генерируя широковещательные сообщения, доставляемые по внутренней сети многопроцессорной системы.

Было придумано множество протоколов когерентности, отличающиеся алгоритмами, количеством состояний и, как следствие скоростью работы и масштабируемостью. Большинство современных протоколов когерентности представляют вариации протокола MESI

В данной схеме каждая линия кэша может находиться в одном из четырех состояний:

* Модифицированная (M) *англ.* modified. Таким флагом может быть помечена только линия в одном кэше. Означает, что данная линия была изменена, но до памяти эти изменения еще не дошли. Хозяин такой линии может спокойно читать и писать в нее без опроса остальных.
* Эксклюзивная (E) *англ.* exclusive. Обозначенная таким флагом линия, так же, как и M-линия может находиться только в одном кэше. Содержащиеся в ней данные полностью идентичны данным в оперативной памяти. Записывать и читать из нее можно без внешних запросов, так как она хранится только в одном кэше. После записи такая строка должна быть помечена, как модифицированная.
* Разделяемая (S) *англ.* shared. Линия может одновременно содержаться в кэшах нескольких устройств и использоваться совместно. Запросы на запись в такую линию всегда идут на общую шину, что привод к тому, что все линии с таким адресов в остальных кэшах помечаются как недействительные. При этом содержание основной памяти также обновляется. Чтение же из такой линии, не требует ни каких внешних запросов.
* Недействительная (I) *англ.* invalid. Такая линия считается невалидной и попытка прочитать приведет к кэш-промаху. Линия помечается недействительной в случае если она пусти или содержит устаревшую информацию.

# 49. Организация оперативной памяти и методы адресации элементов (CAS, RAS).

# **Организация оперативной памяти**

Оперативная память хранит информацию непосредственно доступную процессору. **Максимальная ёмкость оперативной памяти определяется разрядностью адреса и разрядностью минимально адресуемой единицы**. Так при 32-х разрядном адресе и байтовой адресации максимальный размер ОП составляет 232 байт, т.е. 4Гбайт. Существует несколько вариантов организации оперативной памяти.

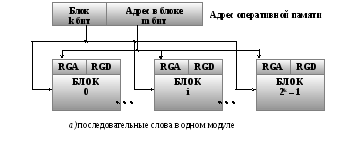
## **Блочная организация оперативной памяти**

Блочная организация оперативной памяти обеспечивает большее быстродействие, наращиваемость и большую живучесть. Классическая интерпретация адреса в этом случае предполагает, что старшие разряды определяют блок, а младшие – слово в блоке, как показано на рис.2 а).

На рис.2 б) приведен пример другого способа интерпретации адреса, когда номер блока указывается младшими разрядами адреса. Такой, более эффективный способ адресации называется *расслоением* или *чередованием адресов*(address interleaving). Если каждый блок является относительно автономным, имеет свой регистр адреса (RGA) и регистр данных (RGD), то, получив адрес и данные, он может выполнять затребованную операцию автономно, а следующее обращение по будет к другому блоку. В идеале такая память может принять столько запросов, сколько блоков. **Реальная эффективность такого способа характеризуется коэффициентом расслоения – среднее количество запросов, которое может быть принято на обслуживание одновременно.** Из таблицы 1 видно, что с ростом количества блоков абсолютный коэффициент расслоения растет, а относительный –снижается и для универсальных ЭВМ стремится к величине https://studfile.net/html/4018/180/html_lhEei1RBsr.nhMj/img-foOs3o.png0.4. Режим расслоения обращений позволяет использовать более медленную ОП. Например, при 8-ми блоках ОП может быть примерно в 3.5 раза медленнее источника запросов на обслуживание.

Таблица 1. Значения коэффициента расслоения.

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во блоков *N* | 2 | 4 | 8 |
| Коэф. рассл. *Кр* | 1.2 – 1.3 | 1.7 – 2.2 | 3.4 – 4.2 |
| https://studfile.net/html/4018/180/html_lhEei1RBsr.nhMj/img-W5xnCL.png | 0.6 – 0.65 | 0.425 – 0.55 | 0.425 – 0.525 |



## **Многопортовая оперативная память**

**Стандартное однопортовое ОЗУ (оперативное запоминающее устройство) имеет по одному входу адреса, данных и управления и в каждый момент времени обеспечивает доступ к ячейке памяти только одному устройству.**

**В отличие от стандартной ОП, в многопортовой памяти имеется несколько независимых наборов шин адреса, данных и управления,** гарантирующих одновременный и независимый доступ к ОП нескольким устройствам (естественно при блочной организации). Данное свойство позволяет существенно упростить создание многопроцессорных и многомашинных вычислительных систем, где многопортовая ОП используется совместно. В рамках одной ЭВМ подобное ЗУ может обеспечивать обмен информацией между ЦП и периферийными устройствами (например, контроллером магнитного диска) намного эффективней, чем прямой доступ к памяти. **Особенностью многопортовой памяти является наличие арбитра запросов.** Конфликты возникают при одновременном обращении через разные порты к одному и тому же блоку. Если же обращения осуществляются к разным блокам, то ОП обслуживает их одновременно.

## **Ассоциативные зу**

В рассмотренных ранее запоминающих устройствах поиск информации осуществляется по адресу ячейки. Другим способом является поиск по признаку (тегу), характеризующему хранимую информацию–ассоциативный поиск. Ассоциативный поиск может быть реализован программно в обычном адресном ЗУ. В ассоциативном ЗУ (АЗУ) ассоциативный поиск реализован аппаратно. Структура ассоциативного ЗУ приведена на рис.3.

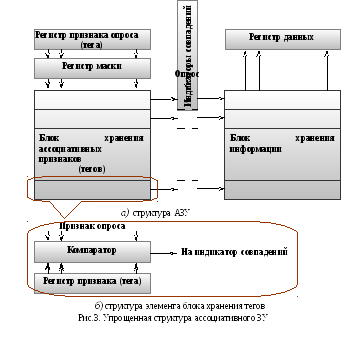
Регистр маски позволяет исключить из поиска определённые биты. На рисунке не показаны средства, обеспечивающие запись. Особенностями АЗУ являются:

* бόльшая емкость за счет блока хранения тегов;
* возможность многозначного ответа;
* возможность организовать сложный поиск.

Ёмкость АЗУ не должна превышать 2n информационных слов, где n–разрядность тега.

Многозначный ответ–это случай, когда несколько ассоциативных признаков совпадает с признаком опроса. В зависимости от области применения АЗУ подобная ситуация обрабатывается дополнительными средствами, например, с помощью регистра маски или является ошибкой, недопустимой.

Сложный поиск имеет целью решение общей задачи поиска - найти слова ассоциативные признаки (теги) которых удовлетворяют заданной комбинации логических отношений к признаку опроса (максимален, минимален, ближайший больший, ближайший меньший, больше заданного, меньше заданного, лежит внутри или вне заданных пределов). Одной из основ сложного поиска является упорядоченная выборка–выборка в порядке возрастания (или убывания) численного значения ассоциативных признаков (тегов).

Существенные отличия в архитектурах АЗУ могут быть связаны с выбранным принципом записи информации. **Запись может осуществляться несколькими способами: по признаку (как чтение), по адресу, в первую свободную ячейку, запись с сортировкой по величине ассоциативного признака.** В последнем случае местоположение ячейки, куда будет помещено новое слово, зависит от соотношения ассоциативных признаков вновь записываемого слова и уже хранящихся в АЗУ. Из-за относительно высокой стоимости АЗУ редко используется как самостоятельный вид памяти.

**За счет чего ускорить доступ к Оперативной памяти?** За счет Конвейризации. Доступ организуется в конвейерном типе, адресуется байт, чтение байта, пока не считается слово. Параллельное выполнение всех ступеней обработки в пределах одного и того же процесса.

**Как соотносятся Risc и Sisd** риск представитель класса сисд архитектуры.

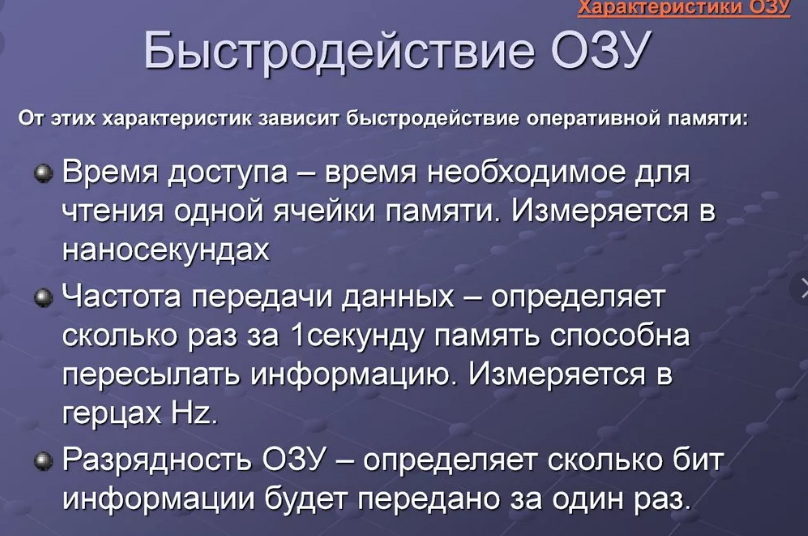
**Что произойдет с кэшем если увеличится объем кэшируемой памяти**

С ростом длины блока при постоянной емкости кэша количество строк кэша уменьшается, и разрядность индекса также снижается, размер поля ’’номер байта в блоке” растет, а объем признакового поля не изменяется.

**Какие поля есть в кэше (в строке кэша)?** Поля: признак, блок данных из памяти, бит грязи, бит достоверности, бит актуальности

**Бит грязи** говорит о том, была ли изменена строка или нет.

**Чем определяется быстродействие оперативной памяти?** Тип, частота, объем.



Быстродействие памяти определяется продолжительностью операции обращения, т.е. временем, затрачиваемым на поиск нужной информации в памяти и на её считывание, или временем на поиск места в памяти, предназначаемого для хранения информации, и на её запись

Методы повышения быстродействия ЦП: повышение тактовой частоты благодаря суперконвейеризации и увеличение степени параллелизма (количества одновременно выполняемых операций в течение каждого периода тактовой частоты) благодаря механизму суперскалярного исполнения

**В чем определяется быстродействие шинного интерфейса?** Разрядностью шины и тактовой чистотой

**Что такое когерентность?** Когерентность кэша — свойства кэш-памяти, означающее целостность данных, хранящихся в локальных кэшах, разделяемой системы.

**Прием для быстрого выбора команд**

Кэширование и ускорение работы памяти, которая в конвейерном виде позволяет считывать байты одного слова.

**От чего зависит пропускная способность шины**

Измеряется в килобайтах (мегабайтах) в секунду и определяется ее тактовой частотой и разрядностью (шириной) шины данных.

Разрядность шины данных определяется разрядностью процессора, т.е. количеством двоичных разрядов, которые процессор обрабатывает за один такт.

**От чего зависит объем кэшированной памяти?** От разрядности признака памяти кэша

**Для чего нужна векторная обработка?**

Для обработки циклов и итераций.

**На уровне какого параллелизма используется многофункциональная обработка**

На уровне операторов и команд

**Имеется кэш, емкость которого увеличилась, что изменилось**

Увеличилось кол-во строк.

**Увеличилась ассоциативность в 2 раза что изменилось?**

С ростом ассоциативности кэша разрядность признакового поля растет, а разрядность индекса падает.

**Что произойдет с кэшем если увеличится объем кэшируемой памяти**

С ростом длины блока при постоянной емкости кэша количество строк кэша уменьшается, и разрядность индекса также снижается, размер поля ’’номер байта в блоке” растет, а объем признакового поля не изменяется.

**Что будет происходить при увеличении длины кэшируемой строки**

Разрядность индекса уменьшится, размер поля увеличится, размер признака увеличится.

**Емкость оперативной памяти возросла, что изменилось в архитектуре кэша?** Признаковое поле возросло добавился еще один разряд.

**На базе каких устройств битовая ячейка** Транзистор и конденсатор

В конденсаторе хранится бит оперативной памяти

Транзистор используется для ограничения cилы тока, поступающего в нагрузку, и включается в разрыв между источником питания и нагрузкой

**Как определяется претендент на вытеснение в случае не обнаружения объекта в кэше? (бит частоты использования блока)**

При организации с прямым отображением Выбирать просто нечего: на попадание проверяется только один блок и только он может быть замещен. При полностью ассоциативной или множественно-ассоциативной организации кэш-памяти имеются несколько блоков, из которых надо выбрать кандидата в случае промаха. Для замещения блоков применяются две основных стратегии: случайная и LRU. В первом случае, чтобы иметь равномерное распределение, блоки-кандидаты выбираются случайно. Во втором случае, чтобы уменьшить вероятность выбрасывания информации, которая скоро может потребоваться, все обращения к блокам фиксируются. Заменяется тот блок, который не использовался дольше всех (LRU – Least-Recently Used).

**Сколько в группе строк кэша?**

Столько же, чему равен коэффициент ассоциативности(А). А = 1 множественный ассоциативный кэш превращается в кэш прямого отображения, а при А, равном числу строк в кэше, - в полностью ассоциативный кэш. кэш прямого отображения содержит  столько групп, сколько блоков в кэше. Соответственно с ростом  ассоциативности кэша размерность признакового поля растет, а  разрядность индекса падает

**Достоинства и недостатки раздельного кеша?**

+выборка команд и данных может производиться одновременно, при этом исключаются возможные конфликты.

- низкая вероятность попаданий по сравнению с совместным

**Какая архитектура самая эффективная?**

Я думаю, что RISC-архитектура - Архитектура процессора, в которой быстродействие  увеличивается ​за счёт ​УПРОЩЕНИЯ ИНСТРУКЦИЙ​, чтобы их  декодирование было более простым, а ​время ​выполнения — меньшим​. Любой CISC-процессор​ УСТУПАЕТ ​RISC-процессорам​ по количеству выполняемых операций в секунду. **Принципы выбора архитектуры.**

Из всех возможных вариантов построения системы лучшим является тот, который обеспечивается наиболее простой архитектурой.

1. **Производительность компьютера** определяется главным образом двумя характеристиками **– тактовой частотой и разрядностью обработки**. Конвейерная организация процессора “работает” на усиление значимости тактовой частоты и является необходимым атрибутом современных компьютеров.

2. Для высокопроизводительных вычислений целесообразно применять RISC-процессоры или процессоры с RISC-ядром. Системы на их основе специально ориентированы на повышение тактовой частоты обработки и “прозрачность” параллелизма архитектуры для прикладного программиста.

3. Предпочтительней использовать один быстрый однопроцессорный компьютер, чем много медленных или многопроцессорную систему с маломощными процессорами.

4. Проблему создания необходимого количества рабочих мест лучше решать с помощью многопользовательских систем на базе сервера и терминалов, чем с помощью ЛВС персональных компьютеров.

5. Вычислительные системы с массовым параллелизмом следует использовать только при полной уверенности в реально существующем параллелизме приложений и только при возможности привлечения программистов высшей квалификации.

**Отличие SMP от MPP**

SMP (symmetric multiprocessing) – симметричная многопроцессорная архитектура. Главной особенностью систем с архитектурой SMP является наличие общей физической памяти, разделяемой всеми процессорами.

MPP (massive parallel processing) – массивно-параллельная архитектура. Главная особенность такой архитектуры состоит в том, что память физически разделена.

Отсутствие общей памяти в MPP архитектуре значительно снижает скорость межпроцессорного обмена, в отличие от SMP архитектуры, в которой общая память присутствует.

Каждый процессор в MPP архитектуре может использовать только ограниченный объем банка памяти, в то время, как процессор в SMP архитектуре охватывает полный объем.

В следствии недостатков MPP архитектура трудно реализуема, что приводит к значительным затратам, реализация SMP архитектуры, напротив, больших затрат не требует.

Главным преимуществом систем с раздельной памятью является хорошая масштабируемость: 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.

**Представители вычислителей класса MIMD**

**МIМD - архитектура** включает все уровни параллелизма от конвейера операций до независимых заданий и программ. Любая вычислительная система этого класса в частных приложениях может выступать как SISD (одиночный поток команд и данных) или SIMD (одиночный поток команд и множество потоков данных) - система.

**Прием неупорядоченной обработки команд**

Позволяет процессору “выдавать" и “ обрабатывать“ команды также не в порядке их следования по программе. Для реализации данной обработки требуется **буфер команд** между ступенями декодирования и исполнения конвейера. Этот прием используется для повышения производительности за счет увеличения числа исполняемых команд за такт.

**Превращение CISC архитектуры в RISC**

Основа успеха микропроцессора на массовом рынке — сохранение совместимости с существующими и будущими приложениями. Для x86 это трудная задачи из-за сложной системы команд, особенностей регистров системного уровня и механизмов защиты памяти и механизмов обработки внешних событий. Решением стало преобразованием x86 команд во внутренние RISC команды.

Главная проблема декодирования CISC команд заключается в определении границ команд и в определении наиболее эффективной последовательности выполнения дешифратором. AMD решили проблему путем распознания сложных команд до стадии декодирования. После определения сложных команд создаются биты предварительного декодирования. Каждый байт в кэш-памяти имеет биты данных декодирования с адресами соответствующих RISC-подобных операций. Процесс предварительного декодирования, происходящий при выборке команд из оперативной памяти в кэш команд, практически не снижает производительности системы, но требует дополнительной кэш-памяти.

**Что делать для того, чтобы снизить время выполнения фаз конвейера?**

Производительность может понижаться из-за:

1) Взаимозависимость по выходу или “запись после записи” возникает при выполнении двух, следующим друг за другом команд, которые записываются в 1 и тот же регистр. Процессор при этом должен гарантировать правильность модификации регистра, даже если команды, модифицирующие его, выполняются не по порядку;

2) Анти-взаимозависимость, или взаимозависимость “запись после чтения“ возникает, когда вторая команда может испортить данные, необходимые в качестве входных для первой.

**Факторы, нарушающие непрерывность конвейерной реализации команд:**

1) Когда для выполнения следующей команды требуется результат действий, реализованных предыдущей командой, или, когда предыдущей командой определяется операнд следующей команды, следовательно, возникает задержка выполнения след. команды.

2) когда в кэш-памяти отсутствуют требуемые данные или команды, необходимо передать их в кэш-память из основной памяти;

3) когда предшествующая команда изменяет содержание следующей или когда изменяется содержимое регистра состояния программы, задающего область выполнения программы, следующая команда должна ожидать завершения предшествующей команды;

4) в случае возникновения прерывания и перехода к программе его обработки (конвейер загружается командами обработки прерывания)

5) когда операция, реализуемая машинной командой, имеет сложный характер и требует для выполнения много машинных циклов, последующая команда долго не может достичь стадии выполнения операции.

**Что такое VLIW?**

**VLIW** – архитектура с очень длинным командным словом. На этапе компиляции несвязанные операции группируются в пакеты, содержимое которых строго соответствует структуре процессора. Таким образом, в рамках данной архитектуры компилятор находит в исходном коде инструкции, выполнять которые можно параллельно, и создает машинный код, использующий этот параллелизм.

**Скалярный процессор** – процессор с единственным конвейером.

**Суперскалярный процессор** – процессор с несколькими конвейерами. Двух потоковые – dual-issue. Четырех потоковые – quad-issue.

**Кластеры**

Кластеры – разновидность MPP-систем.

Кластер состоит из двух или более узлов, удовлетворяющих следующим требованиям:

1. Каждый узел работает со своей копией ОС.

2. Каждый узел работает со своей копией приложения.

3. Узлы делят общий пул других ресурсов (в кластерах приложений). Отдельные экземпляры кластера должны знать о работе друг друга.

**Блок вычислений с плавающей точкой**

Операции с плавающей точкой имеют высокую трудоемкость, поэтому блоки вычислений с плавающей точкой реализуются в виде многоступенчатых конвейеров. SIMD-реализации обработки векторных данных.

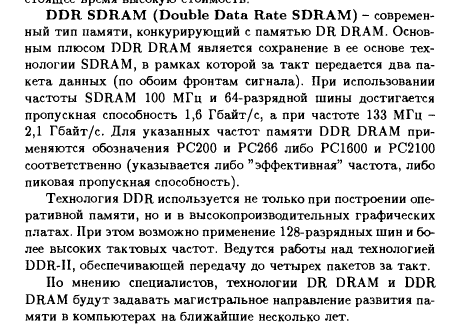
**Кэш память микропроцессора**

Реализация на кристалле раздельного кэша данных и команд. С ростом размера кристалла растет число брака.

50. Синхронный и асинхронный принципы работы оперативной памяти.

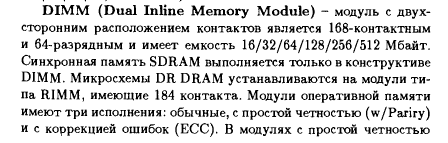
51. SDR, DDR, DDR2, DDR3 памятью. C126-131

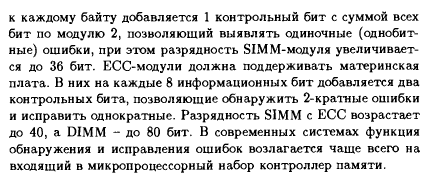
DDR:



52. Регистровая и буферизируемая память (registered DIMM, FB-DIMM).

C 131:





53. Методы обеспечения отказоустойчивости оперативной памяти.

54. Физическая организация дисковой памяти.

55. Физическая и логическая адресация дисковой памяти.

56. Методы обеспечения отказоустойчивости физической памяти, уровни RAID.

**Уровни RAID**

На уровне дисковой подсистемы предпринимаются технические меры, направленные на снижение потерь до стоимости вышедшего из строя оборудования. В основе таких мер лежит резервирование и избыточность, реализуемые в виде массива независимых резервных дисков —**RAID.**

Всего выделяют 9 уровней RAID-массивов, различающихся по скорости, надежности и стоимости изготовления.

**RAID 0** — дисковый массив без дополнительной отказоустойчивости

**RAID 1** – дисковый массив с зеркалированием данных

**RAID 2** — дисковый массив с использованием алгоритма Хэмминга для проверки/восстановления данных

**RAID 3** — дисковый массив с вычислением контрольной суммы параллельно с передачей данных

**RAID 4** — дисковый массив с независимыми дисками данных и общим диском для хранения контрольных сумм

**RAID 5** — дисковый массив с независимыми дисками данных и равномерным распределением контрольных сумм между дисками

**RAID 6** — дисковый массив с независимыми дисками данных и двумя независимыми схемами контрольных сумм, распределенными между дисками

**RAID 7** — дисковый массив с асинхронным вводом-выводом и высокой скоростью передачи данных

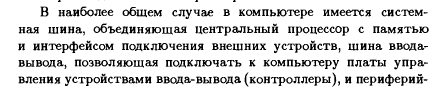
**RAID 10** — комбинация технологий RAID 1 и RAID 0 в одном дисковом массиве

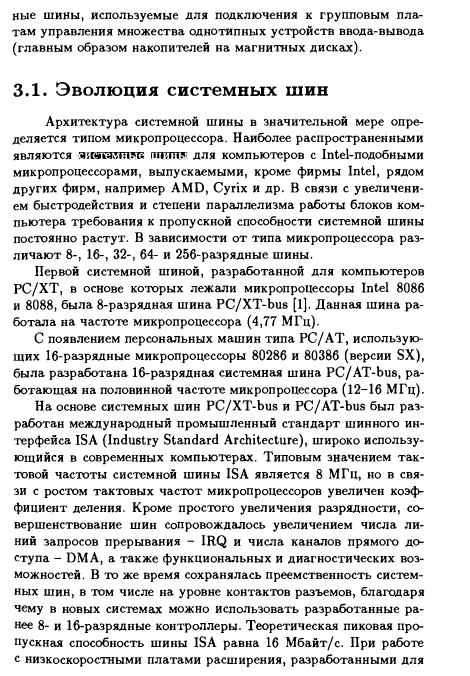
**RAID 53** — комбинация технологий RAID 0 и RAID З в одном дисковом массиве

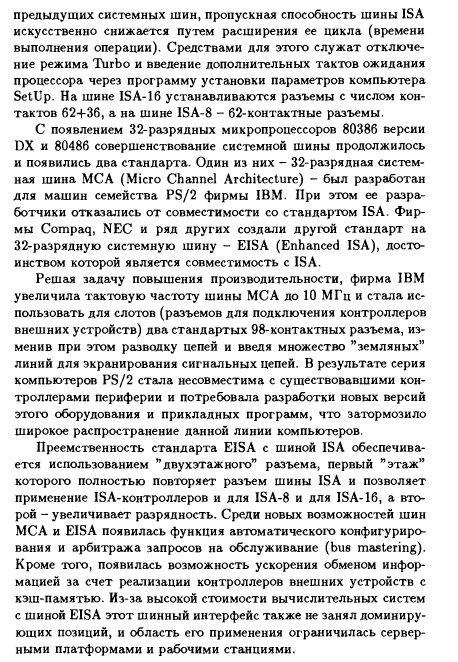
57. Классификация шин по логическим и физическим топологиям.

58. Классификация шин по функциональному назначению.

59. Системные шины. C77-80







60. Шины ввода/вывода.

**ШИНЫ ВВОДА/ВЫВОДА**

В современных вычислительных системах шину ввода-вывода называют также шиной РАСШИРЕНИЯ, поскольку она позволяет подключить к компьютеру широкую номенклатуру дополнительных периферийных устройств.

61. Периферийные шины.

**ПЕРИФЕРИЙНЫЕ ШИНЫ**

Периферийные шины являются компонентами следующего уровня и располагаются между интерфейсными платами контроллеров и периферийными устройствами. Эти интерфейсы изменяются так же быстро, как и остальная часть вычислительной системы

**Различают шину данных, шину адреса, шину управления и состояния. Шиной называют аппаратный стандарт, на основе которого выполняется управление связями между блоками компьютера**

62. Виртуализация: определение и методы.

63. Аппаратная виртуализация.

64. Виртуализация средствами ОС.

65. Виртуализация средствами гипервизоров.

66. Виртуализация серверных групп и ЦОД с использованием решения vSphere от компа-нии VM-ware

5. Зависимость длины конвейера и тактовой частоты процессора.

6. Явления, оказывающие негативное влияние на конвейеризацию.

7. Суперконвейеризация, гиперковейеризация.

Суперскалярная архитектура и методы ее построения.

10. Предикативное и спекулятивное исполнение инструкций

# 13. Неупорядоченное исполнение.

2 файл

**Архитектура вычислительных систем**

# 1. Однопроцессорные архитектуры

Архитектура - это средства превращения программного параллелизма в производительность. Процессор по очереди выбирает команды программы и также по очереди обрабатывает данные.

**1)SISD-архитектура**

Архитектуры данного класса предполагают выполнение только одного потока команд с последовательной обработкой единственного потока данных - SISD (Single Instruction Single Data, один поток команд - один поток данных).

В настоящее время имеется три воплощения SISD-архитектур:

- Coraplete Instruction Set Computing (CISC) - архитектура вычислений с полной системой команд;

- Reduced Instruction Set Computing (RISC) - архитектура вычислений с сокращенным набором команд;

- суперскалярная архитектура, реализующая многофункциональный параллелизм.

**CISC :** Компьютеры с CISC (Complex Instruction Set Computer) архитектурой имеют комплексную (полную) систему команд, под управлением которой выполняются всевозможные операции типа «память-память», «память-регистр», «регистр — память», «регистр — регистр».

Данная архитектура характеризуется:

- большим числом команд (более 200);

- переменной длиной команд (от 1 до 11 байт);

- значительным числом способов адресации и форматов команд;

- сложностью команд и многотактностыо их выполнения;

- наличием микропрограммного управления, что снижает быстродей­ствие и усложняет процессор.

Обмен с памятью в процессе выполнения команды делает практически невозможной глубокую конвейеризацию арифметики, т.е. ограничивается тактовая частота процессора, а значит, и его производительность.

**RISC :** Подразумевает обработку данных, хранящихся только в регистрах. Это дает преимущество в производительности перед CISC-архитектурой, но ценой включения в программу дополнительных команд обмена регистров процессора с оперативной памятью. Отсюда следует, что RISC архитектура производительнее в тех приложениях, где над каждой единицей данных выполняется большой объем вычислительной работы (инженерные и научные задачи), в противном случае часто приходится загружать данные из памяти.

**Суперскалярная архитектура :** базируется на многофункциональном параллелизме и имеет средства, позволяющие одновременно выполнять две или более скалярные операции. Это дает возможность увеличить производительность компьютера пропорционально числу одновременно выполняемых операций. Существует два способа реализации суперскалярной обработки: динамический и статический.

*Динамический способ* - реализуется в архитектуре процессора с предсказанием ветвлений и заключается в чисто аппаратном механизме выборки из ”кэша команд” несвязанных инструкций и их параллельном запуске на исполнение, например, как это имеет место в процессорах DEC серии Alpha, семействе процессоров HP РА-8х00, Intel Pentium-Pro и т.д. Основной проблемой динамической суперскалярной архитектуры является аппаратное предсказание ветвлений.

*Статический способ* реализации суперскалярной обработки основан на VLIW-архитектуре (Very Large Instruction Word) - архитектуре с очень длинным (широким) командным словом.

**Многопотоковая архитектура** основана на введении множества устройств выборки команд на процессорном кристалле, каждое из которых имеет окно выполнения декодированных операций одного потока. Данная и следующая архитектуры фактически переводят SISD-архитектуру процессорного кристалла в класс MIMD архитектур.

**Многоядерная архитектура** реализует несколько полноценных процессорных ядер на одном кристалле [38, 40, 47, 53, 54, 60, 62, 98, 103]. Каждое ядро при этом имеет все атрибуты традиционного процессора - набор исполнительных устройств, регистровый файл, устройство предсказания переходов, индивидуальный кэш первого уровня данных и команд, логику многопотоковой обработки и др. Общим для всех процессорных ядер одного кристалла является, как правило, только кэш второго уровня.

**2) SIMD –архитектура (один поток команд - много потоков данных)**

Различают два принципа реализации векторной обработки: матричный и векторно-конвейерный. **Матричный принцип** - построение системы из четырех квадрантов по 64 процессорных элемента (ПЭ) и 64 модуля локальной памяти при каждом ПЭ, объединенных коммутатором на основе сети типа гиперкуб. Все ПЭ квадранта обрабатывают одну векторную инструкцию, которую им направляет процессор команд.

**Векторно-конвейерный принцип обработки данных** - здесь применяется единственный конвейер операций, имеющий один вход, по которому поступают операнды, и один выход результата, тогда как в матричных системах существует множество входов по данным в процессорные элементы и множество выходов из них.

# 2. Многопроцессорные архитектуры

Многопроцессорные системы могут быть представлены двумя базовыми типами архитектуры в зависимости от параллелизма данных - MISD (Multiple Instruction Single Data, множество потоков команд - один поток данных) и MIMD (Multiple Instruction Multiple Data, множество потоков команд - множество потоков данных).

**MIMD-архитектура** включает все уровни параллелизма от конвейера операций до независимых заданий и программ. Поэтому любая вычислительная система этого класса в частных приложениях может выступать как SISD или SIMD-система.

Чтобы использовать все возможности MIMD архитектуры, необходимо загрузить множество процессоров системы работой в виде множества вычислительных процессов. Существует ’’дерево” многопроцессорных систем. Различают современные высокопроизводительные многопроцессорные вычислительные системы, построенные на основе MIMD-архитектур с сильной (SMP-вычислители) и со слабой (МРР-вычислители) связью процессоров с оперативной памятью вычислительной системы.

SMP -системы. Вычислительные системы с сильной связью ("истинные мультипроцессоры” ), называемые также системами с симметричной мультипроцессорностью, - основаны на объединении процессоров на общем поле памяти, которые совместно с интерфейсной шиной или коммутатором’’ процессоры-память” при увеличении числа процессоров становится узким местом системы.

МРР-системы. Вычислительные системы со слабой связью состоят из узлов (процессор, память, система ввода вывода), работающих под управлением своей копии операционной системы на своем уникальном адресном пространстве. Узлы объединяются между собой коммутационными сетями регулярной структуры (решетка, гиперкуб, тор). МРР-системы называют ’’системами с управлением потоком данных” ("потоковые машины” ) или "системы с архитектурой, отличной от архитектуры фон Неймана” .

# 3. Принципы организации CISC- и RISС- процессоров

Лидером в разработке микропроцессоров c полным набором команд (CISC - Complete Instruction Set Computer) считается компания Intel со своей серией x86 и Pentium. Эта архитектура является практическим стандартом для рынка микрокомпьютеров. Для CISC-процессоров характерно: сравнительно небольшое число регистров общего назначения; большое количество машинных команд, некоторые из которых нагружены семантически аналогично операторам высокоуровневых языков программирования и выполняются за много тактов; большое количество методов адресации; большое количество форматов команд различной разрядности; преобладание двухадресного формата команд; наличие команд обработки типа регистр-память.

*4 основных принципа RISC- архитектуры*.

1. Любая операция, вне зависимости от ее типа, должна выполняться за один такт.
2. Система команд должна содержать минимальное количество наиболее часто используемых простейших инструкций одинаковой длины с минимумом адресных форматов.
3. Операции обработки данных реализуются только в формате "регистр-регистр” . Обмен ’’регистр-память” выполняется только с помощью команд ”загрузки-записи” .
4. Состав системы команд должен быть ”удобен” для компиляции операторов языков высокого уровня.

Специфика организации RISC-процессоров во многом определяется "целевой функцией” их базовой архитектуры, которая предполагает достижение возможного максимума производительности путем использования мощных средств обработки. Это достигается тремя структурными приемами:

1. Полноразрядной обработкой данных (каждый арифметический конвейер за каждый так т формирует один 64-разрядный результат, что является оптимальным форматом данных с плавающей и фиксированной точкой для вычислительных задач).
2. Конвейерной организацией арифметических и других функциональных исполнительных устройств.
3. Суперскалярной структурой процессора с комплексом средств динамического прогнозирования ветвлений, большим ко­личеством регистров (минимум 32 регистра общего назначения, выполняющих роль сверхоперативной памяти), многоуровневым кэшем.

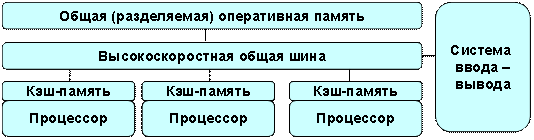
# 4. Принципы выбора архитектуры

1. Производительность компьютера определяется главным образом двумя характеристиками - тактовой частотой и разрядностью обработки. Конвейерная организация процессора "работает” на усиление значимости тактовой частоты и является необходимым атрибутом современных компьютеров.
2. Для высокопроизводительных вычислений целесообразно применять RISC-процессоры или процессоры с RISC-ядром. Системы на их основе специально ориентированы на повышение тактовой частоты обработки и ”прозрачность” параллелизма архитектуры для прикладного программиста.
3. Предпочтительней использовать один быстрый однопроцессорный компьютер, чем много медленных или многопроцессорную систему с маломощными процессорами.
4. Проблему создания необходимого количества рабочих мест лучше решать с помощью многопользовательских систем на базе сервера и терминалов, чем с помощью ЛВС персональных компьютеров
5. Вычислительные системы с массовым параллелизмом следует использовать только при полной уверенности в реально существующем параллелизме приложений и только при возможности привлечения программистов высшей квалификации.

# 5. Подходы к организации многопроцессорных вычислительных систем

**SMP — симметричная многопроцессорная архитектура**

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



***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*.

**CMP - перестраиваемая симметрично многопроцессорная архитектура**

Это перестраиваемая SMP-архитектура, которая позволяет комбинировать SMP и кластерные технологии. Данная архитектура многопроцессорных серверных платформ предложена компанией Unisys. В основе СМР лежит модель однородного общего поля оперативной! памяти. Строительные блоки в СМР - процессорные элементы (Unisys subpods), ориентированы на применение процессоров Intel Pentium Xeon и Mersed в рамках одной СМР-системы.

Процедуры, основанные на каталогах. В архитектуру СМР заложены уникальные возможности по статическому и динамическому парционированию (разбиению) SMP-сервера, приводящие к преобразованию в кластер всей SMP-системы, построенной из SMP-серверов с числом процессоров, кратным четырем.  
В СМР возможно три типа разделения поля оперативной памяти между разделами:

- каждая ОС использует только свою память;

- каждая ОС имеет свою память и образуется еще одна общая для разных ОС область памяти;

- каждая ОС имеет свою память и образуется несколько областей памяти, разделяемых некоторыми ОС.

**МРР - многопроцессорная архитектура с распределенной памятью**

Многопроцессорные вычислительные системы с распределенной (индивидуальной) памятью строятся на основе МРР архитектуры и относятся к классу NUMA. Узлы в архитектуре МРР обычно состоят из одного процессора, памяти и устройств ввода-вывода. В каждом узле работает своя копия ОС, а узлы объединяются между собой специализированным соединением.

**Кластеры - разновидность МРР-систем**

В области компьютерных технологий понятие ’’кластер” применяют в двух значениях. Различают вычислительные кластеры и кластеры, используемые для повышения надежности и отказоустойчивости серверов приложений (кластеры приложений). С содержательной точки зрения кластеры имеют МРР-архитектуру с более дешевой, чем у вычислителей с массовым параллелизмом, коммуникационной подсистемой. В этом смысле кластеры являются одним из направлений развития компьютеров с массовым параллелизмом.

Кластер состоит из двух или более узлов, удовлетворяющих следующим требованиям:

- каждый узел работает со своей копией ОС;

- каждый узел работает со своей копией приложения;

- узлы делят общий пул других ресурсов (в кластерах приложений).

В отличие от кластера приложений в МРР-системах узлы не делят ресурсы для хранения. Это главное отличие между кластерными SMP-системами и традиционными МРР-системами.

Архитектура CC -NUMA данная архитектура является развитием SMP-архитектур и представляет собой совокупность многопроцессорных SMP-узлов с собственной памятью, объединенных между собой скоростной общей шиной либо коммутатором. Объединение узлов выполняется через оперативную память. Каждый процессор имеет собственную локальную память и может устанавливать статические и динамические соединения с модулями памяти других процессоров. По существу, CC-NUMA - это кэш-когерентный доступ к неоднородной памяти. В системе CC-NUMA физически распределенная память объединяется в единый массив [13, 34, 35, 89]. Существует одна карта памяти (единое адресное пространство) с частями, физически связанными специальной соединительной шиной, и аппаратно-реализованная кэш-когерентность.

# 6. Доменная архитектура многопроцессорных вычислительных систем

Доменная архитектура основана на концепции разбиения SMP вычислителей, предусматривающей сегментирование ресурсов вычислительной системы и создание изолированных друг от друга разделов, обеспечивающих независимую работу разных приложений. Данная архитектура позволяет объединить в одном вычислителе несколько полнофункциональных независимых серверных систем, каждая из которых имеет свои процессоры, память, подсистему ввода-вывода и экземпляр операционной системы.

**Системные разделы**

Системное разбиение позволяет создать на базе одного аппаратного сервера несколько разделов, каждый из которых представляет собой полнофункциональную серверную систему, работающую под управлением собственного экземпляра ОС. Системное разбиение является необходимой функциональностью для провайдеров приложений. Системный раздел включает все необходимые для автономной работы ОС ресурсы - процессоры, память, устройства ввода вывода.

**Разделение приложений**

Разделение приложений является удобным механизмом управления ресурсами вычислительной системы при предоставлении информационных услуг. Возможность закрепления за конкретным приложением или группой пользователей необходимых ресурсов упрощает достижение соглашений об уровне обслуживания (QoS). При разделении приложений наборы ресурсов для рабочих областей приложений, контролируемых одним экземпляром ОС, выделяются в ’’ прикладные разделы”. Разделение приложений дополняет системное разбиение и допустимо только в определенных аппаратных границах; на одном автономном сервере, в серверном кластере, в одном системном разделе. Следующий уровень программного разбиения обеспечивает ”планирование классов” - сегментирование ресурсов прикладного раздела на несколько прикладных или пользовательских классов и явное выделение части ресурсов группе пользователей или приложений, входящих в класс. В основе разделения приложений лежит выделение некоторого числа процессорных блоков на сервере или системном разде­ле в процессорный набор.

# Архитектурные приемы

Простейший фактор, влияющий на производительность цп – его **тактовая частота**. Основной путь повышения производительности - построение архитектур, выполняющих за **такт** более одной команды.

**Способы увеличения производительности**:

1. Конвейеризация
2. Суперскалярные архитектуры
3. Неупорядоченное выполнение
4. Переименование регистров
5. Обходы и продвижение данных
6. Прогнозирование переходов
7. Превращение CISC-архитектуры в RISC
8. Блоки вычислений с плавающей точкой

## Конвейеризация

В большинстве классических CISC-архитектур **5 ступеней обработки** конвейера:

1) ВЫБОРКА КОМАНДЫ

2) ДЕКОДИРОВАНИЕ КОМАНДЫ

3) АДРЕСАЦИЯ И ВЫБОРКА ОПЕРАНДОВ

4) ИСПОЛНЕНИЕ КОМАНДЫ

5) ЗАПИСЬ РЕЗУЛЬТАТА

Конвейерная обработка представляет собой **параллельное выполнение всех ступеней обработки** в пределах одного и того же процесса. Каждая ступень конвейера обрабатывает свою фазу команды **независимо.** В каждый момент времени конвейерный процессор работает над выполнением различных стадий нескольких команд.

В общем случае различные команды процессора имеют неодинаковые длительности выполнения различных фаз.

Конвейерное выполнение команд имеет **максимальную эффективность**, когда продолжительность выполнения всех этапов команд одинакова, бесперебойно подаются команды и данные, на каждом этапе отсутствуют зоны ожидания выполнения аналогичного этапа очередной команды.

*Что нарушает непрерывность конвейерной реализации команд:*

1. Когда для выполнения следующей команды требуется результат действий, реализованных предыдущей командой.  
    Или когда предыдущей командой определяется адрес операнда следующей команды (модификация адреса). Тогда возникает **задержка начала выполнения следующей команды**, связанная с ожиданием выборки операнда следующей команды или с преобразованием адресов;
2. При ветвлении программы по результатам проверки условий командой условного перехода команды, находящиеся в процессе конвейерной обработки, остаются невыполненными, и требуется повторная загрузка конвейера, начиная с момента выборки команды условного перехода;
3. Когда в кэш-памяти отсутствуют требуемые данные или команды, необходимо передать их в кэш-память из основной памяти;
4. Когда предшествующая команда изменяет содержание последующей.  
   Или когда изменяется содержимое регистра состояния программы, задающего область выполнения программы, последующая команда должна ожидать завершения предшествующей команды;
5. В случае возникновения прерывания и перехода к программе его обработки команды, находящиеся в это время на командном конвейере, остаются незавершенными и приходится заново загружать конвейер командами, входящими в программу обработки прерывания;
6. Когда операция, реализуемая машинной командой, имеет сложный характер и требует для выполнения много машинных циклов, последующая команда долго не может достичь стадии выполнения операции. В современных процессорах пять основных ступеней делятся на многоступенные операции, позволяющие снизить сложность каждой ступени и обеспечить однородность конвейера, - такая организация вычислений называется **суперконвейерной**.

Идея суперконвейера заключается в сокращении числа логических операций, которые необходимо выполнить на каждой ступени конвейера с тем, чтобы можно было быстрее переходить со ступени на ступень и, следовательно, повысить тактовую частоту.

## 1.2 Суперскалярные архитектуры

Процессор *с единственным конвейером* называется **скалярным**. Процессоры с *несколькими конвейерами* называются **суперскалярными** (многопотоковыми).

Как правило, микропроцессоры имеют ограничения на типы команд, обрабатываемых одновременно. **Существует два подхода к отображению параллелизма обработки данных** на архитектурном уровне в системе команд:

1. Система команд не содержит внутри процессора никакого указания на параллельную обработку. В таких процессорах параллелизм исполнения команд формируется динамически аппаратурой микропроцессора.
2. В специальных полях команды каждому из параллельных исполнительных устройств указывается действие, которое устройство должно совершить. Это **процессоры с длинным командным словом** (VLIW). Загрузка параллельных исполнительных устройств этих микропроцессоров формируется специальными компиляторами с языков высокого уровня.

**Основная идея** суперскалярных микропроцессоров, состоит в том, что компиляторы и логика микропроцессора сами, без вмешательства программиста, обеспечивают загрузку параллельно работающих устройств микропроцессора.

## 1.3 Неупорядоченное выполнение

В конвейерах возможны **заторы (зависания)**, обусловленные взаимозависимостями команд в программе и способами построения суперскалярных архитектур.

1. Существуют архитектуры, использующие **«упорядоченное поступление и обработку»** и **«упорядоченное завершение».** В таких архитектурах все, что затрудняет завершение команды в одном конвейере, останавливает и другой конвейер, т. к. команды должны покидать конвейеры точно в том же порядке, в каком поступали на них. *Это упрощает конструкцию процессора, но увеличивает производительность.*
2. Архитектуры с **«неупорядоченным завершением»** и **«неупорядоченным исполнением»** позволяют одному из конвейеров продолжать работать при «заторе» в другом. При этом команды, стоящие в программе позже, могут быть выполнены раньше предыдущих команд, «застрявших» в другом конвейере. *Но процессор должен гарантировать, что результаты не будут записаны в память, а регистры модифицироваться в неправильной последовательности.*
3. **«Неупорядоченная обработка»** развивает предыдущую концепцию дальше, позволяя процессору «выдавать» и «обрабатывать» команды также не в порядке их следования по программе. Для реализации неупорядоченной обработки обычно требуется *буфер команд (или «окно команд»)* между ступенями декодирования и исполнения конвейера.

## 1.4 Переименование регистров

Производительность процессора ограничена из-за того, что некоторые операции над связанными данными принципиально нельзя завершить до выполнения других.

1. В выражении **A x B + C** сложение и умножение нельзя выполнить одновременно согласно правилам арифметики. Это **«истинная взаимозависимость данных»**, или взаимозависимостью **«чтение после записи»**, т.е. входные данные для одной операции зависят от результата другой.
2. Производительность может снижаться, кроме того, **«ложными взаимозависимостями»** двух видов;
   1. **Взаимозависимость по выходу**, или взамозависимость **«запись после записи»** возникает при выполнении двух, следующих друг за другом команд, которые записывают свои результаты в один и тот же регистр;
   2. **Антивзаимозависимость,** или взаимозависимость **«запись после чтения»** возникает, когда вторая команда может испортить данные, необходимые в качестве входных для первой.

Частое использование одних регистров приводит к возникновению взаимозависи­мостей и снижению производительности. *Путь повышения производительности - нейтрализация ложных взаимозависимостей с помощью переименования регистров.*

* **RISC-процессоры** имеют большое количество регистров и **ложные взаимозависимости возникают редко.**
* **CISC-процессоры**, ориентированные на широкое разнообразие методов адресации, оснащены небольшим числом регистров, что провоцирует **частое возникновение ложных взаимозависимостей**.

Если команде требуется регистр, процессор динамически ставит в соответствие этому логическому (архитектурному) регистру один из множества физических регистров. Если другая команда пытается обратиться к тому же логическому регистру, процессор для предотвращения конфликта может поставить ему в соответствие другой физический регистр. Такие переименования действуют, пока команды продвигаются по конвейерам.

## 1.5 Обходы и продвижение данных

Переименование регистров не может совсем исключить возникновение ИСТИННЫХ взаимозависимостей данных. Для нейтрализации таких взаимозависимостей используются методы **ОБХОДА ДАННЫХ** и **ПРОДВИЖЕНИЯ ДАННЫХ**.

1. При ОБХОДАХ результаты выполнения одной команды сразу пересылаются следующей и таким образом исключаются задержки на модификацию и повторное чтение из регистра или памяти.
2. При ПРОДВИЖЕНИИ ДАННЫХ процессор может выполнять некоторые команды параллельно, немедленно передавая результаты одной из них в другую, которой они потребуются на более поздней ступени конвейерной обработки. При этом также исключается ожидание завершения операций записи и чтения адресуемых операндов.

## **1.6 Прогнозирование переходов**

*Суперскалярность, неупорядоченная обработка, продвижение данных* могут реализовать свой потенциал только при разрешении проблемы **процедурных взаимозависимостей**.

**Причина возникновения**: из-за наличия условных и безусловных переходов в программе (изменения последовательности выполнения команд). Т. к. *условные переходы* (ветвления) разрешаются только на исполнительной ступени конвейера, процессор не знает, какие команды следует направлять в конвейер за командой перехода.   
Здесь используется «предположение о пути ветвления» - **прогнозирование ветвления**. После прогноза процессор начинает:

* выборку команд с предсказанного адреса перехода до того, как он узнает, верным ли был его прогноз;
* исполнение по предположению «спекулятивное»; при этом процессор не может модифицировать архитектурные регистры или память до уточнения достоверности предсказанного перехода.

Некоторые процессоры обеспечивают несколько уровней предположений, прогнозируя дополнительные ветвления до разрешения первого.

*Приемы прогнозирования ветвлений:*

1. Использование **буфера адреса перехода** на 256 и более позиций, который отслеживает и хранит данные о результатах 256 последних ветвлений.

* Каждая строка буферной таблицы указателей перехода **включает адрес команды**, задаваемый командой перехода, **адрес указателя направления перехода** команды перехода и **предысторию команды перехода**.
* **Предыстория** содержит информацию о выполнении или невыполнении условий перехода данной команды до текущего момента.
* Во время **декодирования команды** перехода по предыстории прогнозируется выполнение или невыполнение условий команды перехода и производятся вызов и декодирование команд из прогнозируемой ветви программы.

1. Буфер адресов переходов дополняется **стеком возвратов**, который отслеживает переходы в паре команд CALL/RETURN.
2. Информация о переходах добавляется к каждой строке кэша команд.

* Здесь дается адрес первого перехода в строке кэшпамяти и указывается, как следует прогнозировать ветвление - как *выполняемое* или *невыполняемое*.
* Недостаток в том, что нельзя сохранить информацию о более чем одном переходе в строке кэша.

## 1.7 Превращение CISC-архитектуры в RISC

Один из подходов повышения быстродействия микропроцессора и обеспечения преемственности основан на RISC-ядре и схемах преобразования команд CISС во внутренние RISC-команды.

**Главная проблема декодирования** **CISC-команд**: определение *границ команд* и расположение их в такой последовательности, которая обеспечит наиболее эффективную обработку дешифратором, т. к. быстрое декодирование команд является необходимым условием для обработки нескольких команд за такт.

**Эффективное решение проблемы декодирования** состоит в распознавании сложных команд х86 до стадии декодирования на этапе загрузки команд из основной памяти в кэш-память команд. Это **предварительное декодирование** сопровождается формированием битов предварительного декодирования. При этом каждый байт в кэш-памяти команд имеет биты данных декодирования, указывающих начало и конец команды и число основных RISC-подобных операций, необходимых для ее выполнения.

Процесс предварительного декодирования требует дополнительной кэш-памяти. При выборке команд из кэша границы команд уже определены и RISC-преобразователи формируют серии RISC-команд.

## 1.8 Блоки вычислений с плавающей точкой

**Операции с плавающей точкой имеют высокую трудоемкость**. В связи с этим блоки вычислений с плавающей точкой в современных микропроцессорах реализуются в виде **многоступенчатых конвейеров**, позволяющих увеличить тактовую частоту исполнительных устройств.

Еще одним способом увеличения скорости выполнения операций с плавающей точкой является SIMD-реализация обработки векторных данных. В настоящее время параллельная обработка данных с плавающей точкой с помощью SIMD-расширений базового набора команд становится необходимым атрибутом практически всех микропроцессоров от различных производителей.

## 1.9 Кэш-память микропроцессора

Для сокращения потерь, связанных с доступом к внекристальной памяти, в составе микропроцессора имеется **внутренний кэш первого уровня** наиболее популярной гарвардской архитектуры *с разделением кэш-памяти на отдельные секции команд и данных.*

**Преимущество:** Возможность динамического разделения при работе программ большого объема или обрабатывающих большие массивы данных.

**Недостаток:** Возникновение конфликтов между доступом к коду и данным на различных стадиях конвейерной обработки. Поэтому чаще всего на первом уровне иерархической памяти в кристалле микропроцессора реализуется *раздельный кэш для данных и команд*.

Важнейшим фактором, определяющим быстродействие вычислительной системы, является **емкость кэша**. Объем кэша первого уровня современных микропроцессоров колеблется от 32 Кбайт до 1,5 или 2,25 Мбайт.

## 1.10 Многопотоковые и многоядерные микропроцессоры

* **Многопотоковые микропроцессоры** позволяют на основе одного набора исполнительных функциональных устройств орга­низовать *исполнение нескольких независимых потоков команд*. Данный подход направлен на увеличение загрузки устройств выполнения команд, которые в традиционных архитектурах загружены на реальных приложениях в среднем на 20-30%.
* Многопотоковые микропроцессоры требуют реализации для каждого потока блоков выборки команд и переименования регистров, механизмов неупорядоченного выполнения команд и прогнозирования переходов. При этом накладные расходы на организацию вычислений в каждом отдельном потоке, как правило, не превышают 5-10%.
* **Многоядерные микропроцессоры** содержат на кристалле несколько самостоятельных процессорных ядер. Это позволяет реализовать все рассмотренные выше архитектурные приемы увеличения быстродействия процессоров, в том числе многопотоковую архитектуру.

# **Тенденции развития микропроцессоров**

Основные усилия архитекторов микропроцессоров сосредоточены на следующих направлениях:

* Повышение тактовой частоты;
* Увеличение объема и пропускной способности подсистемы памяти;
* Увеличение количества параллельно работающих функциональных исполнительных устройств;
* Введение блоков обработки мультимедийных данных;
* Интеграция на кристалле функций управления памятью и периферийными устройствами, для исполнения которых в традиционных микропроцессорах используются наборы микросхем («чипсеты»);
* Интеграция на кристалле интерфейсов сетевых и телекоммуникационных систем, позволяющая соединять эти микропроцессоры друг с другом и телекоммуникационными и вычислительными сетями без дополнительных адаптеров;
* Создание многопроцессорных и многопотоковых (многонитевых) систем на одном кристалле, обеспечивающих распараллеливание на уровне процессов и потоков (нитей).

# **Архитектура микропроцессора PentiumPro (Р6)**

## 3.1 Общая характеристика

Повышение производительности микропроцессорной вычислительной системы возможно за счет:

* разработки высокопроизводительных компонент, окружающих микропроцессор;
* увеличения кэш-памяти 2-го уровня;
* повышения частоты микропроцессора;
* совершенствования микроархитектуры микропроцессора.

В архитектуре Р6 реализованы следующие способы увеличения производительности:

* суперскалярность (до 3-х команд х86 одновременно);
* суперконвейерность (14-ступенчатый конвейер);
* выполнение команд с изменением последовательности;
* предсказание переходов;
* исполнение по предположению;
* переименование регистров;
* преобразование команд х86 в RISC-подобные микрокоманды;
* кэш первого и второго уровней неблокирующего типа;
* кэш второго уровня
* на частоте микропроцессора;
* шина кэша второго уровня отделена от шины оперативной памяти;
* использование шины транзакций, допускающей одновременное выполнение операций ввода-вывода и обращений к оперативной памяти.

Весь набор архитектурных приемов, использованных в Р6, получил общее название **«Динамическое исполнение команд»**.

В Р6 использованы **2** основных *метода повышения быстродействия ЦП*:

1. повышение тактовой частоты благодаря суперконвейеризации
2. увеличение степени параллелизма (количества одновременно выполняемых операций в течение каждого периода тактовой частоты) благодаря механизму суперскалярного исполнения.

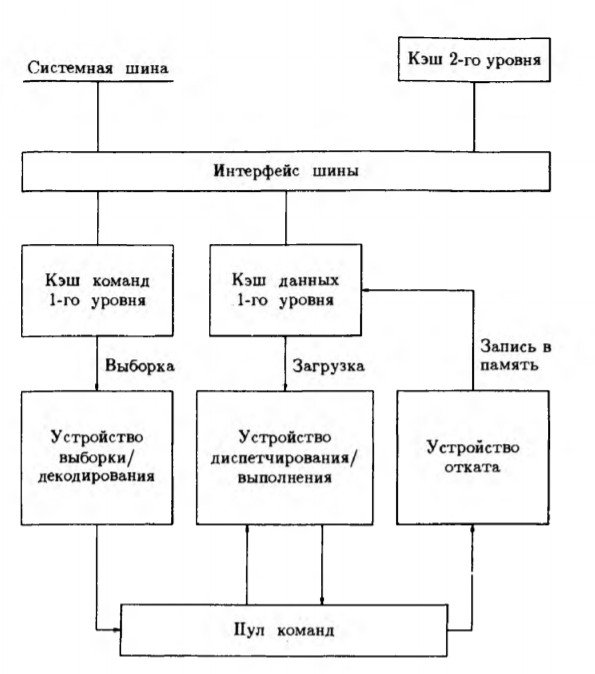
## 3.2 Архитектура суперконвейера

Упорядоченное устройство – устройство, которое работает в соответствии с исходным порядком команд в программе.

Беспорядочное устройство – устройство, которое не обращает внимания на исходный порядок команд в программе.

Конвейер Р6 имеет **14 ступеней (фаз)**, разделенных на **три независимых блока (устройства)**, взаимодействующих через пул команд:

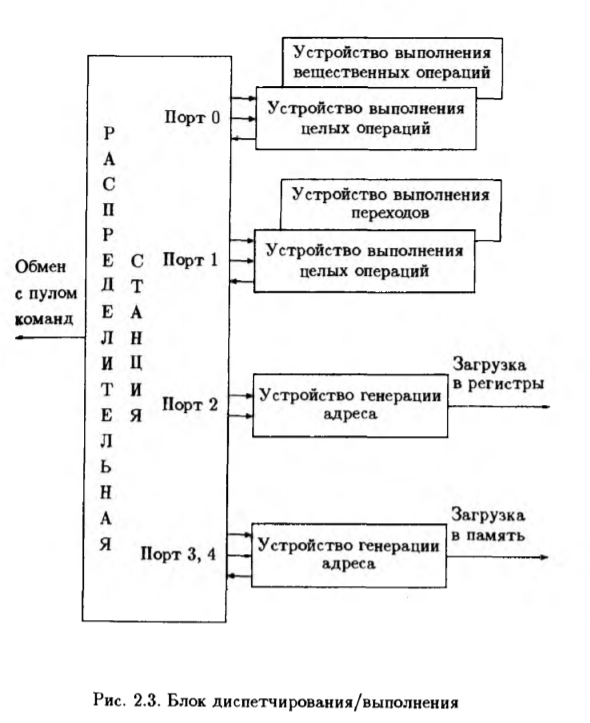
1. **входной блок выборки/декодирования команд**, состоящий из 8 ступеней и являющийся ” упорядоченным” устройством, работающим в соответствии с исходным порядком команд в коде программы;
2. **блок диспетчирования/выполнения команд** с изменением последовательности, где происходит собственно выполнение команд; имеет **3 ступени** и является ’’ беспорядочным” устройством, которое не обращает внимания на исходный порядок команд в программе, а планирует их выполнение с учетом зависимостей по данным и доступности ресурсов, вре­менно сохраняя результаты опережающего выполнения в пуле команд;
3. **блок отката (вывода) команд из последовательности**, состоящий из **3 ступеней** - ’’упорядоченное” устройство, отвечающее за перевод временных результатов опережающего выполнения в постоянное состояние вычислительной системы.

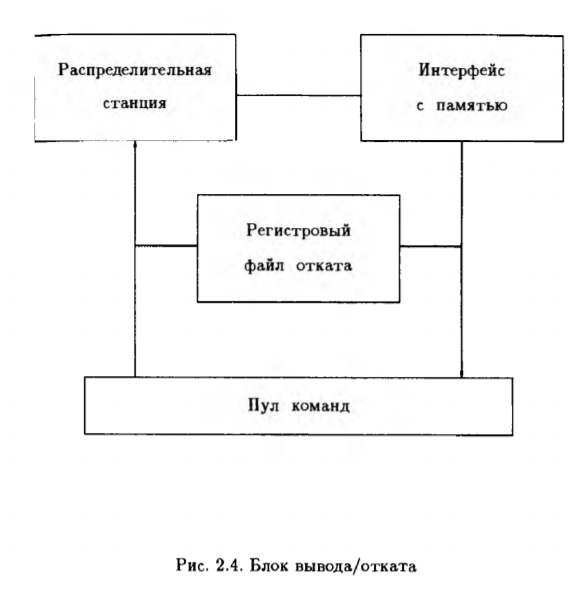


**Интерфейс шины** является ’’частично упорядоченным” устройством, отвечающим за связь 3 блоков суперконвейера с внешним миром. Интерфейс шины взаимодействует непосредственно с кэшем второго уровня и поддерживает до четырех параллельных обращений к кэшу.



**Блок выборки/декодирования** реализует функции определения указателя на следующую команду, выравнивания команд перед посылкой на дешифраторы, трансляции команд х86 (1-6-я ступени конвейера), отображения регистров (7-я, 8-я ступени конвейера). Преобразование команд х86 в RISC-подобные микрокоманды позволяет устранить ограничения набора команд х86, связанные с нерегулярностью кодирования команд, переменной длиной непосредственных операндов и частыми пересылками регистр-память. Выборка кода и трансляция его в микрокоманды происходит на первых шести ступенях конвейера Р6.





## **3.3 Предсказание переходов**

По оценкам экспертов, в типичной программе до 10% команд могут быть безусловными переходами и еще до 10-20% - условными переходами. При ошибке в предсказании перехода Р6 может потерять от 4 до 15 тактов.

Существует два основных метода предсказания переходов - **статический и динамический.**

1. **Статические методы** предписывают всегда выполнять или не выполнять определенные типы переходов.
2. **Динамические методы** оценивают поведение команд перехода за предшествующий период времени, поскольку один и тот же переход часто выполняется более чем один раз, например в цикле.

Р6 применяет статические методы для предсказания поведения команд переходов, предыстория которых не была проанализирована с помощью динамических методов.   
В Р6 используется структура - буфер адреса перехода (branch target buffer - ВТВ) и предсказания, основанные на битах предыстории ветвлений.

# 3. Шинные интерфейсы

**Интерфейс** это совокупность аппаратных, программных и конструктивных средств и правил, обеспечивающих взаимодействие компонентов вычислительной системы или сети и учитывающих информационную, электрическую и конструктивную совместимость их элементов.

**Протокол** определяет правила взаимодействия элементов вычислительной системы и способ выполнения определенных функций.

**Канал** определяет среду распространения сигналов.

**Шина** это совокупность линий интерфейса, сгруппированных по функциональному назначению. Различают **шину данных, шину адреса, шину управления и состояния.**

Таким образом, шиной называют аппаратный стандарт, на основе которого выполняется управление связями между блоками компьютера. **Пропускная способность** любой шины измеряется в **килобайтах (мегабайтах) в секунду** и определяется ее **тактовой частотой и разрядностью (шириной)** шины данных.

**Пиковая скорость передачи** данных по шине определяется произведением этих параметров.

В общем случае в компьютере имеется **системная шина**, объединяющая цп с памятью и интерфейсом подключения внешних устройств, шина ввода-вывода, позволяющая подключать к компьютеру платы управления устройствами ввода-вывода (контроллеры), и периферий­ные шины, используемые для подключения к групповым платам управления множества однотипных устройств ввода-вывода.

## Шины ввода-вывода

Шину ввода-вывода называют также **шиной расширения**, поскольку она позволяет подключить к компьютеру широкую номенклатуру дополнительных периферийных устройств. Шинные интерфейсы ввода-вывода используются **только** в качестве шины для подключения контроллеров внешней периферии.

### Шина PCI и унаследованные шины

Шина PCI стала в настоящее время стандартом подключения внешних устройств для широкого разнообразия вычислительных архитектур, предлагаемых различными производителями. Для настольных систем реализуется обычно 32-разрядная версия стандарта, работающая на частоте 33 МГц, а для серверных платформ и высокопроизводительных рабочих станций используется четыре версии; 32 бит/33 МГц, 32 бит/66 МГц, 64 бит/33 МГц и самая скоростная 64 бит/66 МГц.

В настоящее время интерфейс PCI принят в качестве стандарта на шину ввода-вывода для вычислительных платформ на базе процессоров Alpha фирмы DEC и SPARC компании Sun Microsystems.

Для подключения большого числа контроллеров периферии, разработанных в стандарте ISA (EISA), по-прежнему широко используется данный шинный интерфейс. Однако самые современные материнские платы для старших моделей микропроцессоров уже не содержат слотов расширения в стандарте ISA.

### Ускоренный графический порт AGP

С ростом производительности вычислительных систем происходит дальнейшее расслоение шинных интерфейсов - так для управления видеоподсистемой современного компьютера используется специальный ускоренный графический порт - Accelerated Graphics Port (AGP), являющийся самостоятельной шиной обмена графической информацией.

Порт AGP имеет самостоятельный доступ к памяти и позволяет разгрузить шину PCI для обслуживания других быстродействующих устройств. Разрядность шины AGP составляет 32 бит, рабочая частота - 66 МГц, в шине поддерживаются режимы передачи х1, х2, х4. В этих режимах за один такт передаются соответственно одно, два или четыре 32-разрядных слова, а пиковая производительность в режиме х1 - 266 Мбайт/с, х2 - 533 Мбайт/с и х4 - 1066 Мбайт/с.

В общем случае существует несколько основных способов повышения скорости отображения графической информации:

* ускорение передачи данных из основной памяти, передача большего числа функций генерации изображения видеоконтроллеру, повышение тактовой частоты контроллера,
* увеличение разряд­ности локальной шины между видеоконтроллером и расположенной на видеокарте памятью,
* применение быстродействующей видеопамяти,
* а также увеличение объема этой памяти для кэширования данных.

### Интерфейс ввода-вывода на основе коммутатора

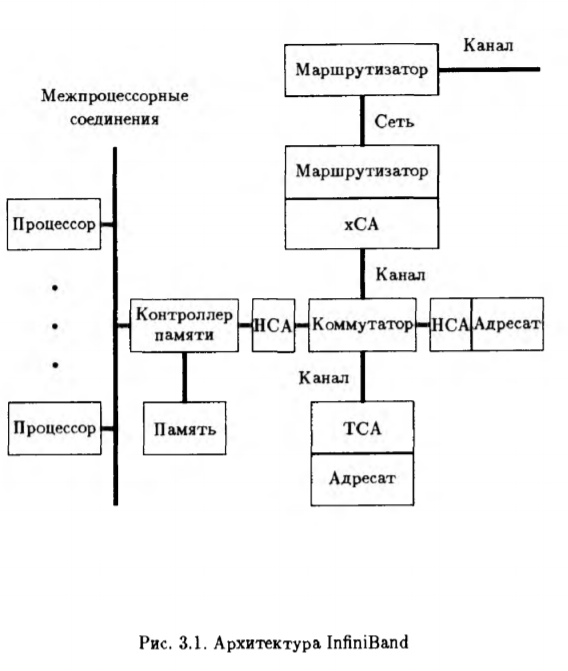
Наряду с шиной PCI-X появились две принципиально новые архитектуры локальных шин - **Future I/O (FIO)** и **Next Generation I/O (NGIO).**

В отличие от разделяемой архитектуры шины PCI, характеризующейся совместным использованием шины всеми подключенными устройствами, Future I /O опирается на коммутируемую структуру или матрицу с прямыми соединениями между устройствами. Как и в любой коммутируемой среде, подключение очередного устройства не сказывается на доступной любому из них пропускной способности, а наоборот ведет к увеличению совокупной пропускной способности, так как каждое из устройств имеет свое отдельное соединение. Первоначально пропускная способность соединений Future I/O будет составлять 1 Гбайт в каждом направлении. Предполагается, что Future I/O будет применяться для межсоединений процессоров в кластерах, подсистем и процессоров в сетях хранения, а также для подключения высокоскоростной периферии и сетей на базе Fibre Channel, TJltraS SCSI и Gigabit Ethernet.

Архитектура шины Next Generation I /O так же, как и Future I/O , основывается на коммутируемой технологии. Кроме того, NGIO использует канальную архитектуру, реализующую эффективный механизм ввода-вывода для обработки запросов от периферии. Канальная архитектура NGIO включает главный канальный адаптер - Host Channel Adapter (НСА), интерфейс с контроллером памяти вычислителя и целевые канальные адаптеры - Target Channel Adapter (ТСА). Главный канальный адаптер содержит механизмы прямого доступа к памяти. Целевые канальные адаптеры служат для подключения контроллеров ввода-вывода к коммутатору. НСА и ТСА могут подключаться либо к другу.

### Архитектура ввода-вывода InfiniBand

Стандарт InfiniBand описывает архитектуру и спецификации на передачу данных между процессорами и интеллектуальными устройствами ввода-вывода. Принципиальное отличие InfiniBand от PCI заключается в замене общей шины коммутаторами, что обеспечивает широкие возможности кластеризации и масштабирования вычислительных систем.



## 3.3. Периферийные шины

**Периферийные шины** являются компонентами следующего уровня и располагаются между интерфейсными платами контроллеров и периферийными устройствами. Эти интерфейсы изменяются так же быстро, как и остальная часть вычислительной системы.

### Шина EIDE

Простейшей современной шиной *для подключения устройств долговременной памяти* является расширенный интерфейс подключения к компьютерам АТ накопителей со встроенным контроллером - АТ Attachment/Enhanced Integrated Drive Electronics (ATA/EIDE).

Эта шина имеет **два канала** (Primery и Secondary), к каждому из которых можно подключить по **два устройства** (Master и Slave).

Теоретическая пропускная способность составляет 16,7 Мбайт/с.

**Длина** интерфейсного кабеля до 45 см.

**Недостатком** является то, что значительную часть работы по управлению устройствами в стандарте EIDE выполняет процессор. Кроме того, для управления дисками большой емкости используются неэффективные методы.

Шина EIDE используется в основном в настольных персональных компьютерах.

### Семейство шин SCSI

С точки зрения максимальной теоретической пропускной способности и числа подключаемых периферийных устройств ведущее положение среди периферийных шин в настоящее время занимает интерфейс малых вычислительных систем - Small Computer Systems Interface (SCSI). Данный интерфейс допускает реализацию в контроллере шины (адаптере) до трех 8- или 16-разрядных каналов, называемых узкими (Narrow) или широкими (Wide) соответственно, и позволяет подключать до 7 или 15 устройств к каждому каналу в зависимости от его разрядности.

Имеется несколько спецификаций шины, совместимых снизу вверх, - то есть при поддержке контроллером и подключенным устройством разных модификаций стандарта использоваться будет менее скоростная.

Шина может находиться в одном из следующих состояний (фаз):

* свободна (BUS FREE);
* выбор (SELECTION);
* передача команд (COMMAND);
* передача статуса (STATUS);
* передача сообщения (MESSAGE);
* арбитраж (ARBITRATION);
* повторный выбор (RESELECTION).

Терминирование бывает двух типов: пассивное и активное.

### Шина IEEE-1394

IEEE-1394, FireWire и i.LINK - это три названия высокоскоростного интерфейса последовательной шины стандарта SCSI. Последовательный дешевый аналог параллельных шин.

Данный интерфейс обладает всеми достоинствами (преимуществами) действующего интерфейса SCSI и пропускной способностью 100-400 Мбит/с.

Кроме того, предусмотрены пути модернизации шины до гигабитных скоростей. Шина поддерживается фирмой Apple Computers и разработана на базе оригинальной шины FireWire, предложенной Apple Computers.

**Область применения** IEEEM-1394 - высокоскоростной доступ к устройствам хранения информации (жесткие диски, приводы CD и DVD), устройствам ввода данных (сканеры), аудио- и видеооборудованию.

Стандарт обладает гибкостью и простотой использования, обеспечивает приоритетную передачу, подключение и отключение устройств в ’’ горячем режиме” (без выключения питания и перезагрузки), способен конфигурироваться в автоматическом режиме.

### Шина Fibre Channel

**Fibre Channel** - комплект протоколов, определяющий высокую скорость передачи данных (до 1 Гбайт/с) между компьютерами, периферией и другими системами через волоконнооптический интерфейс. Является конкурентом SCSI.

Fibre Channel чаще всего используется в сетях хранения данных - Storage Area Network (SAN). Первоначально стандарт основывался на использовании только оптоволоконной среды передачи, однако в процессе разработки в спецификацию был включен и кабель с медными жилами.

Fibre Channel представляет собой высокопроизводительный последовательный канал ’’точка-точка” как самостоятельно работающий, так и позволяющий работать поверх него протоколам SCSI, FDDI (протокол построения локальных сетей) и другим.

В стандарте предусмотрены коммутаторы, обеспечивающие соединения ’’точка-многоточка”, что делает его пригодным для применения в локальных сетях высокой пропускной способности.

### Универсальная последовательная шина USB

**Universal Serial Bus (USB)** - универсальная последовательная шина - новый стандарт с пропускной способностью 12 Мбит/с для подключения низко- и среднескоростных устройств типа: клавиатуры, мыши, джойстика, микрофона, принтера, модема, дисковода CD-ROM и т.д.

Общее число подключаемых устройств - до 127. Поддерживается фирмами Intel, Microsoft Corp.

Данный интерфейс позволяет снять проблему недостаточного количества прерываний, каналов DMA, адресов ввода-вывода при подключении большого числа медленных устройств. В ближайшие годы по мере роста числа производимых периферийных устройств с портами USB произойдет полный переход на данный тип интерфейса в персональных компьютерах. При этом классические последовательные и параллельные порты, порты для подключения клавиатуры и мыши будут вытеснены.

Кроме того, с внедрением USB будет вытеснена и шина ввода-вывода ISA, через которую реализовывалось подключение низко- и среднескоростных устройств.

Архитектурой USB предусматривается топология **"звезда”**. Система должна состоять из одного ведущего (корневого) концентратора с контроллером, управляемым операционной системой, требуемого количества концентраторов и узлов (периферийных устройств). Концентраторы могут каскадироваться, образуя древовидную структуру с поддеревьями. Узлы подключаются к концентраторам. Всего узлов может быть 127. Концентратор также считается устройством.

При построении цепочек дерева наиболее скоростные устройства (монитор и т.п.) следует подключать ближе к корневому концентратору, а наименее скоростные (клавиатура и т.п.) - в конце цепочки. Это обеспечивает постепенное снижение интенсивности трафика от корня к концу цепочки и приоритетное обслуживание быстродействующих устройств.

# Подсистема памяти

## Архитектура многоуровневой памяти.

Главными требованиями, которым должны удовлетворять подсистема памяти, являются достаточно большая ёмкость, высокое быстродействие и экономическая эффективность с точки зрения технической реализации. При минимальных физических размерах память должна обладать как можно большей информационной ёмкостью.

Различные уровни иерархической памяти работают асинхронно. Для этого существуют буферные устройства, которые позволяют параллельно выполнять операции доступа к различным уровням.

Подсистема памяти имеет многоуровневую иерархическую структуру. На верхнем уровне – регистровая память, на следующем уровне – буферная память (кэш-память), затем оперативная память произвольного доступа, на самом нижнем – внешняя память.

Регистры общего назначения (РОН) – сверхоперативное запоминающее устройство (СОЗУ). Объём СОЗУ обычно составляет несколько десятков регистров. Регистровая память предназначена для хранения адресов, операндов и результатов выполнения операций.

Кэш-память – высокоскоростная память произвольного доступа, используемая процессором компьютера для временного хранения информации. Увеличивает производительность.

Внешняя память располагается на дисковых и ленточных запоминающих устройствах. Память самая медленная, но самая ёмкая.

Есть несколько уровней кэш-памяти (до трёх уровней). Первый (внутренний) расположен на кристалле процессора. Ёмкость колеблется от нескольких десятков килобайт до полутора мегабайт, время доступа – один-два тактов. Второй уровень реализуется либо на кристалле процессора, либо в виде отдельной микросхемы на одной плате, либо как статическая память. Ёмкость – от нескольких сотен килобайт до нескольких мегабайт, время доступа – три-пять тактов. Кэш третьего уровня имеет внешнее исполнение. Ёмкость до нескольких мегабайт, время доступа – до десятков тактов.

## Статическая и динамическая память

Ячейка динамической памяти представляет собой конденсатор на полупроводниковом кристалле. Его зарядка означает переход в состояние 1. Разрядка – в состояние 0. Одна из причин ограниченного быстродействия этого типа памяти – длительная зарядка и разрядка ёмкости, ограниченность во времени хранения заряда конденсатором. Чтобы избежать потери данных, необходимо восстановление в ней информации – недостаток динамической памяти. Достоинства – дешевизна и минимальное потребление мощности.

Скорость заряда конденсатора будет зависить от сопротивлений в цепи и емкости кондера, а сам заряд будет идти по экспоненциальному закону.

Статическая память строится на элементах памяти с двумя устойчивыми состояниями – триггерах. Бит отображается состоянием триггера – электронного ключа, состоящего из четырёх транзисторов и двух резисторов. Переход триггера из одного состояния в другое происходит быстрее, чем зарядка/разрядка. Минусы: потребляет большую мощность, чем динамическая память; значительно дороже и при одинаковой степени интеграции с динамической, обладает меньшей информационной ёмкостью.

## Кэш-память

Кэш используется для обмена между процессором и памятью, а также между оперативной памятью и внешним накопителем. В основе работы кэш-памяти лежит принцип временной и пространственной локальности программ. Принцип пространственной локальности предполагает считывание в кэш нескольких соседних ячеек памяти (блока инф-ии). Каждый блок хранится в строке буфера, набор таких строк составляет кэш-память.

Чем больше размер блока, тем выше коэффициент удачных обращений. С другой стороны, чем больше размер блока, тем меньше их помещается в кэше – растёт число операций пересылки из памяти в кэш и обратно.

Информация из основной памяти загружается в кэш блоками по несколько слов и временно хранится в нём.

# Классификация типов кэша

Различают классификации по способам хранения данных и команд, организации записи данных и организации отображения памяти. Для хранения данных и команд применяют общие и раздельные кэши. Раздельный используется в кэш-памяти первого уровня. Общие кэши – на втором и третьем уровнях иерархической памяти.

Существует два способа обновления информации, гарантирующих адекватность содержимого кэша и основной памяти: сквозная запись и обратная запись. Сквозная запись: информация записывается как в кэш, так и в оперативную память. Обратная запись: сохраняет блок кэша в оперативной памяти только при вытеснении его из кэша и при наличии в нём изменений.

В зависимости от способа отображения различают типы кэшей:

* С прямым отображением
* Полностью ассоциативный
* Множественный ассоциативный

## Кэш с прямым отображением

Является самым простым типом буферной памяти. Кэш состоит из строк, содержащих информационную и признаковую части. Обычно размер блока составляет 32 или 64 байта.

При обращении центрального процессора к адресному объекту, выполняется:

1. Из адреса объекта выделяются три поля:

- номер байта в блоке;

- индекс, он же номер блока в кэше, в котором должен находится адресуемый объект;

- признак, позволяющий отличить один блок оперативной памяти от другого;

1. По индексу, выделенному из адреса объекта, находится строка кэше, в которой должен находиться нужный блок памяти;
2. Признак, выделенный из адреса объекта, сравнивается с признаком, хранящимся в элементе признаковой памяти найденной строки кэша.

Достоинства: необходимость только одного сравнения признака, выбранного из строки и признака, выделенного из адреса объекта. Недостатки: конфликт адресов блоков памяти, отображаемых на одну строку кэша. При возникновении конфликта адресов, старый блок переписывается из кэша в память, а на его место помещается новый.

## Полностью ассоциативный кэш

Любой блок оперативной памяти может быть отображен на любую строку кэша. Так как между блоками нет определённых взаимосвязей, в строку кэша должен записываться полный адрес объекта и сам блок. Этот подход позволяет решить проблему конфликта адресов.

Пока кэш не заполнен, блок помещается в первую свободную строку. Когда буфер полон, один из блоков должен быть из него вытеснен и переписан в основную память, а на его место помещён новый.

## Множественный ассоциативный кэш

Множество строк разбиваются на группы по А≥1 строк. А целочисленное – коэффициент ассоциативности кэша. А = 2 – двухвходовый множественный ассоциативный кэш, А = 4 – четырёхвходовый и т д.

Индекс адресует номер группы и адресуемый блок может располагаться в любой строке группы. Между группами множественный ассоциативный кэш является кэшем прямого отображения, а внутри группы – полностью ассоциативным.

С ростом ассоциативности, с одной стороны, растёт частота «попаданий» в кэш, а с другой – растут и накладные расходы в виде количества сравнений признаков.

При А = 1 множественный ассоциативный кэш превращается в кэш прямого отображения, при А = числу строк – в полностью ассоциативный кэш.

# Принципы организации оперативной памяти

## Элемент динамической памяти

В состав ячейки динамической памяти входит конденсатор и транзистор.

Вот чёто вроде надо написать, а чё, я не ебу

## Массивы ячеек и структура микросхем динамической памяти

Микросхемы динамической памяти организованы в виде массивов строк и столбцов. На все ячейки одной строки приходится общая числовая шина, а на все ячейки одного столбца отводится общая пара разрядных шин, усилитель и буфер обмена. Место каждой ячейки определяется адресом строки и столбца.

Микросхема DRAM имеет матричную организацию, каждый элемент которой хранит бит данных и адресуется с помощью стробирующих сигналов адреса строки – RAS и адреса столбца – CAS. Сигналы RAS и CAS используются также для восстановления данных. Продолжительность процесса восстановления определяется количеством строк матрицы. Соотношение количества строк и столбцов информационного поля DRAM влияет на энергопотребление микросхемы, способ адресации и совместимость с микросхемами различных поколений. Может ещё дописать, нахрена только это

## Многоблочная структура памяти и расслоение адресов

Для ускорения выполнения операций чтения и записи группы последовательных байт оперативная память подразделяется на множество независимых модулей (блоков). Это разбиение позволяет повысить эквивалентное быстродействие памяти с помощью расслоения адресов по независимым блокам.

# Обзор технологий FPM, EDO, SDRAM, DR DRAM, DDR DRAM

FPM DRAM – технология адресации, при которой выбор данных в пределах одной страницы (строки) выполняется с указанием полного адреса страницы и столбца только в момент считывания первого байта, а последующие байты той же страницы считываются без указания её адреса. Сигнал CAS используется для адресации различных ячеек в пределах строки и для задания конца периода доступности адреса, в течение которого выполняется считывание данных.

EDO DRAM – тип памяти с расширенным временем удержания данных выходе за счёт применения элементов конвейеризации. EDO – технология адресации, при которой в качестве сигнала конца операции чтения используется незанятый сигнал OE. Это даёт возможность начать адресацию нового столбца до завершения чтения данных из предыдущего.

SDRAM – технология адресации последовательности слов в пакете, при которой после выборки первого слова не используется адрес текущего столбца. Особенностью этой технологии является зависимость всех управляющих сигналов от общего системного тактового сигнала.

DR DRAM – (новый) тип памяти, основанный на шине Direct Rambus, в которой управление адресацией отделено от работы с данными.

DDR DRAM – тип памяти. Который конкурирует с памятью DR DRAM. Плюсом данной памяти является сохранение в её основе технологии SDRAM, в рамках которой за такт передаётся два пакета данных. Используется не только в оперативной памяти. Но и в высокопроизводительных графических платах.

# Типы модулей памяти

SIMM – модуль с односторонним расположением контактов, имеет исполнение двух видов: 30-контактные 8-разрядные и 72-контактные 32-разрядные.

DIMM – модуль с двухсторонним расположением контактов – 168-контактный и 64-разрядный модуль. Синхронная память SDRAM выполняется только в конструктиве DIMM. Модули оперативной памяти имеют три испольнения: обычные, с простой четностью и с коррекцией ошибок.

# Банки памяти

Оперативная память с точки зрения процессора – совокупность нескольких отдельно адресуемых банков в виде набора SIMM или DIMM модулей памяти, общая разрядность которых соответствует максимальной разрядности шины данных процессора. Один банк памяти в компьютерах на базе процессора Pentium, состоит из двух 32-разрядных SIMM модулей или одного 64-битного DIMM.

В целях минимизации нагрузки и повышения силы выходного тока производится согласование шин данных с помощью специальных буферов-микросхем, устанавливаемых на плате SIMM или DIMM. Модуль с буфером называется buffered, а небуферизованный – unbuffered. Несмотря на преимущество буферизованных модулей памяти, их использование ограничено: необходимо чтобы данный тип модулей памяти поддерживался микропроцессорным набором вычислительной платформы.

При выборе модулей памяти, руководствуются правилом: модуль памяти с большим числом микросхем малой ёмкости нагружает шины сильнее, чем модуль с меньшим числом микросхем большей ёмкости.

# Механизм динамического преобразования адресов:

Реально существующую основную память называют физической, а её адреса – физическими, логическую память – виртуальной, а её адреса – виртуальными.

Обращение к таблице преобразования адресов при преобразовании виртуального адреса в реальный занимает много времени. Для ускорения этой процедуры в вычислительных системах имеется механизм динамического преобразования адресов, основанный на кэш-памяти ассоциативного типа. В эту кэш-память записываются номера наиболее часто используемых страниц и номера блоков, соответствующих этим страницам в основной памяти. При преобразовании адресов сначала проверяется ассоциативный кэш, в случае промаха – основная таблица преобразования, расположенная в оперативной памяти.

# Дисковые запоминающиеся устройства

## Факторы, определяющие производительность

Конструкция головок чтения-записи, запоминающая среда, каналы обмена, скорость вращения, количество дисковых пластин.

Время, необходимое дисковому накопителю для обслуживания запроса пользователя, имеет четыре составляющие: время обработки команды, время поиска дорожки, задержка вращения, время передачи данных.

Время обработки команды зависит от интерфейса диска, типа команды и возможности использования буфера или кэша жёсткого диска.

Важным фактором, определяющим время поиска дорожки, является размер и вес компонентов диска.

Задержка вращения определяется временем ожидания заданного сектора подл магнитными головками.

Время передачи данных зависит от скорости передачи и размера передаваемого блока.

## Влияние плотности записи битов

Плотность битов (линейная плотность) показывает, сколько битов можно сохранить на одной дорожке. От этого зависит число секторов на дорожке. Чем выше линейная плотность, тем больше секторов на дорожке.

Увеличение числа битов на дюйм оказывает влияние на следующие рабочие характеристики диска:

* Возрастает скорость передачи носителя
* Возникают ограничения на скорость вращения
* Уменьшается число переключений головки при переходе на следующую дорожку данного цилиндра и на дорожку следующего цилиндра
* Растёт размер цилиндра, выраженный в количестве секторов, так как увеличивается число секторов на дорожке

## Влияние плотности размещения дорожек

При одинаковом числе секторов на дорожке и одинаковом количестве дорожек в цилиндре более высокая плотность дорожек означает короткую дистанцию физического перемещения головки от одного блока к другому и, следовательно, обеспечивает меньшее время поиска. Для более узких дорожек с более плотным расположением требуется большее время позиционирования головки.

Повышение плотности записи даёт возможность получать диск заданной ёмкости с меньшим числом пластин и меньшим числом головок. Фактор роста плотности записи позволяет снизить стоимость дискового устройства и приводит к снижению установившейся скорости передачи данных размера цилиндров.

# Средства обеспечения отказоустойчивости и масштабируемости

## Отказоустойчивые массивы дисков (RAID)

На уровне дисковой подсистемы предпринимаются технические меры, направленные на снижение потерь до стоимости вышедшего из строя оборудования. В основе таких мер лежит резервирование и избыточность, реализуемые в виде массива независимых резервных дисков (RAID). Существует программная и аппаратная реализация RAID-массива.

RAID-система состоит из управляющей программы, или контроллера, и массива НЖМД. RAID-массив строится на основе распределения данных между отдельными дисками. Пространство каждого диска разбивается на сегменты, размер которых может составлять от одного сектора (512 байт) до нескольких мегабайт. Различные логические формы организации RAID-массива называются уровнями RAID.

Повышение уровня готовности и отказоустойчивости достигается с помощью ряда аппаратных средств:

* Средств изменения конфигурации без отключения сервера
* Возможности наращивания ёмкости системы в процессе работы
* Возможности изменения «на ходу» уровня RAID
* Устройств «горячей замены», горячего резерва или избыточных дисков, контроллеров, блоков питания, вентиляторов охлаждения
* Средств резервного питания кэш-памяти контроллера RAID-массива.

Высокая производительность обеспечивается за счёт кэширования.

## Уровни RAID

Существует девять уровней (типов) RAID-массивов, различающихся по скорости, надёжности и стоимости изготовления: 0, 1, 2, 3, 4, 5, 7, 10, 53.

RAID 0 – дисковый массив без дополнительной отказоустойчивости.

Достоинства:

* Высокая производительность за счёт распределения операций ввода-вывода между всеми дисками массива
* Не производится подсчёт контрольных сумм, что увеличивает производительность
* Простота конструкции и лёгкость в изготовлении

Недостатки:

* Выход из строя одного из дисков приводит к потере всех данных, хранящихся в дисковой подсистеме

RAID 1 – дисковый массив с зеркалированием данных.

Достоинства:

* Скорость записи та же, что и для одного диска
* Скорость чтения в два раза выше, чем для одного диска
* Высокая скорость восстановления данных из-за их стопроцентной избыточности
* Самый простой конструктив из всех типов RAID-массивов
* Единственный тип RAID-массива, позволяющий получить отказоустойчивую дисковую подсистему на двух дисках

Недостатки:

* Низкий коэффициент использования дискового пространства, высокая стоимость.

RAID 2 – дисковый массив с использованием алгоритма Хэмминга для проверки/восстановления данных.

Достоинства:

* Исправление ошибок данных «на лету»
* Высокая скорость передачи данных, увеличивающаяся с ростом количества дисков в массиве
* Коэффициент использования дискового пространства увеличивается с ростом количества дисков в массиве
* Относительно простой конструктив контроллера

Недостатки:

* Очень низкий коэффициент использования дискового пространства в случае малого размера слова данных

RAID 3 – дисковый массив с вычислением контрольной суммы параллельно с передачей данных.

Достоинства:

* Очень высокая скорость чтения и записи данных
* Выход из строя одного диска незначительно влияет на общую производительность массива
* Высокий коэффициент использования дискового пространства

Недостатки:

* Контроллер имеет конструктив средней сложности
* Высокая трудоёмкость программной реализации, требующая значительной вычислительной мощности

RAID 4 – дисковый массив с независимыми дисками данных и общим диском для хранения контрольных сумм.

Достоинства:

* Высокая скорость чтения данных
* Высокий коэффициент использования дискового пространства

Недостатки:

* Наименьшая из всех типов RAID-массивов скорость записи
* Достаточно сложный конструктив контроллера
* Сложный и неэффективный алгоритм восстановления данных при выходе из строя одного из дисков

RAID 5 – дисковый массив с независимыми дисками данных и равномерным распределением контрольных сумм между дисками.

Достоинства:

* Высокая скорость чтения и записи данных
* Высокий коэффициент использования дискового пространства

Недостатки:

* Выход из строя дисковода оказывает заметное влияние на производительность
* Сложный конструктив контроллера
* Сложный алгоритм восстановления данных в случае выхода из строя одного из дисков

RAID 6 – дисковый массив с независимыми дисками данных и двумя независимыми схемами контрольных сумм, распределёнными между дисками.

Достоинства:

* Высокая скорость чтения данных
* Высокий отказоустойчивость

Недостатки:

* Сложный конструктив контроллера
* Большая нагрузка на контроллер при вычислении контрольных сумм и адресов, по которым они должны быть размещены на дисках
* Очень малая скорость записи
* Низкий коэффициент использования дискового пространства

RAID 7 – дисковый массив с асинхронным вводом-выводом и высокой скоростью передачи данных.

Все операции ввода-вывода проводятся в асинхронном режиме, то есть независимо от выполнения других.

Достоинства:

* Общая производительность на операциях записи на 25-90% выше, чем для одного диска, и в 1,5-6 раз лучше, чем у RAID-массивов других типов
* Очень высокая скорость доступа к данным в многопользовательской среде, реализуемая за счёт кэширования
* Скорость чтения и записи растёт при увеличении числа дисков в массиве

Недостатки:

* Высокая стоимость

RAID 10 – комбинация технологий RAID 1 и RAID 0 в одном дисковом массиве.

Достоинства:

* Имеет такую же отказоустойчивость, как и RAID 1
* Скорость чтения и записи выше, чем у RAID 1

Недостатки:

* Высокая стоимость
* Самый низкий коэффициент использования дискового пространства из всех типов RAID-массивов

RAID 53 – комбинация технологий RAID 0 и RAID 3 в одном дисковом массиве.

Достоинства:

* имеет такую же отказоустойчивость, как и RAID 3
* Скорость чтения и записи выше, чем у RAID 3

Недостатки:

* Высокая стоимость
* Низкий коэффициент использования дискового пространства из всех типов RAID-массивов

Централизованные и локальные RAID-массивы

RAID-системы разделяют на:

* Системы с централизованным управлением (host-based) – вся интеллектуальная начинка RAID-массива размещается на плате, устанавливаемой в сервер.
* Локальные системы (SCSI-to-SCSI) – управляющая часть расположена в RAID-стойках, подключаемых к серверу через SCSI-контроллер.

# Кластеры

Кластеризация вычислительных систем – технология, с помощью которой два и более серверов функционируют как один вычислитель, воспринимаемый и клиентскими приложениями, и пользователями как единая вычислительная система.

Используется для обеспечения:

* Отказоустойчивости
* Постоянной доступности
* Масштабируемости, достигаемой реализацией распределения вычислительной нагрузки

Существует три модели кластерных систем:

* Модель с разделяемой памятью
* Модель с разделяемыми дисками
* Модель без разделяемых ресурсов

## 6.1 Модели состоятельности памяти.

При увеличении процессорных мощностей встает вопрос о построении памяти, имеющей единое адресное пространство и доступ к объектам через операции чтения и записи. При этом требуется извещать процессор об изменениях в памяти. Рассмотрим наиболее используемые модели состоятельности памяти, которые решают данные вопросы:

1. Строгая – каждая операция чтения возвращает последнее записанное значение
2. Последовательная – все процессоры в системе соблюдают один и тот же порядок выполнения записи и чтения. *Процессор, выполняющий запись, приостанавливается до получения подтверждений об объявлении несостоятельными всех копий модифицируемых данных или о модификации этих копий.*
3. Процессорная – наблюдаемый в двух процессорах порядок выполнения операций чтения-записи может не совпадать, но порядок выполнения записей, производимых каждым процессором, должен быть одним и тем же.
4. Слабая - вводит разграничение между обычным и синхронизованным доступом в память при выполнении состоятельности только для обращений к памяти в точках синхронизации различных уровней памяти. Разрешается несогласованность данных вне точек синхронизации.
5. Свободная – уточняет модель слабой состоятельности и требует состоятельности только между парой операций синхронизации *(acquire – release),* которые открывают интервал и обеспечивают исключительный доступ процессора к разделяемым данным и закрывают его и разрешают всем процессорам доступ к разделяемым данным. Доступны следующие модификации:
   1. Неторопливо-свободная – модификация разделяемой памяти откладывается до момента непосредственного обращения к ней.
   2. Нетерпеливо-свободная – модель с передачей изменений разделимой памяти при выполнении release, независимо от того, когда изменения будут востребованы.
   3. Интервально-свободная- модель с разбиением на локальные и глобальные интервалы. Гарантирует состоятельность разделяемых данных только в конце глобального интервала
   4. Последовательно-свободная – модель, требующая доступа к разделяемым данным, защищённым синхропримитивами.

## 6.2 Неявная(аппаратная) когерентность.

Применяются аппаратные средства для обеспечения когерентности данных в памяти, например, для кэшей с помощью межузловых пересылок.

## 6.2.1 Сосредоточенная память.

Для систем с сосредоточенной памятью в симметричных многопроцессорных архитектурах применяется алгоритм поддержки когерентности кэшей MESI. Он обеспечивает когерентность кэш-памяти с обратной записью и минимизирует пересылки данных между основной памятью и кэшем.   
Алгоритм предполагает, что вычислитель имеет общую разделяемую память, в каждом процессоре содержится локальная кэш память, а объединены они между собой посредством шины.

(*В современных компьютерах доступ к памяти могут одновременно иметь несколько независимых процессоров (ядер, трэдов). Каждый из них имеет свои приватные кэши, в которых хранятся копии необходимых линий, а некоторые из них при этом локально модифицированы. Встает вопрос, а что, если одна и та же линия одновременно понадобится нескольким процессорам. Не сложно сделать вывод, что для корректной работы системы необходимо обеспечить единое пространство памяти для всех процессоров.  
Для обеспечения этого были придуманы специальные протоколы когерентности. Когерентность кэша — свойства кэш-памяти, означающее целостность данных, хранящихся в локальных кэшах, разделяемой системы. Каждая ячейка кэша имеет флаги, описывающие, как ее состояние соотносится с состоянием ячейки с таким же адресом в других процессорах системы.  
При изменении состояния текущей ячейки необходимо каким-то образом сообщить об этом остальным кэшам. Например, генерируя широковещательных сообщения, доставляемые по внутренней сети многопроцессорной системы.  
Было придумано множество протоколов когерентности, отличающиеся алгоритмами, количеством состояний и, как следствие скоростью работы и масштабируемостью. Большинство современных протоколов когерентности представляют вариации протокола MESI*)

В данной схеме каждая линию кэша может находиться в одном из четырех состояний:  
**М-строка модифицирована** (*Таким флагом может быть помечена только линия в одном кэше. Данное состояние означает, что данная линия была изменена, но до памяти эти изменения еще не дошли. Хозяин такой линии может спокойно читать и писать в нее без опроса остальных.)***E-строка монопольно копированная** (*Обозначенная таким флагом линия, так же, как и M-линия может находиться только в одном кэше. Содержащиеся в ней данные полностью идентичны данным в оперативной памяти. Записывать и читать из нее можно без внешних запросов, так как она хранится только в одном кэше. После записи такая строка должна быть помечена, как модифицированная.)***S – строка множественно копированная или разделяемая.** (Линия *может одновременно содержаться в кэшах нескольких устройств и использоваться совместно. Запросы на запись в такую линию всегда идут на общую шину, что привод к тому, что все линии с таким адресов в остальных кэшах помечаются как недействительные. При этом содержание основной памяти также обновляется. Чтение же из такой линии, не требует никаких внешних запросов.)*  
**I – строка невозможная к использованию.** (*Такая линия считается не валидной, и попытка прочитать приведет к кэш-промаху. Линия помечается недействительной в случае, если она пусти или содержит устаревшую информацию.)*

## 6.2.2 Распределённая память.

ВС с памятью, физически распределенной по узлам(модулям), имеющей единое пространство адресов применяют алгоритмы реализации когерентности.

*Например, ПРОСТЕЙШИЙ заключается в том, что при промахе в кэш в любом процессоре инициируется запрос требуемой строки из резидентного (того, где строка размещена) блока. Запрос передается в узел с резидентным блоком памяти, откуда строка пересылается в узел, где произошёл промах. Так обеспечивается начальное заполнение кэшей.*  
*Для обеспечения когерентности после выполнения записи процессор приостанавливается до завершения этой последовательности действий:  
-измененная строка пересылается в резидентный блок  
- разделяемая строка пересылается из резидентной памяти во все узлы, перечисленные в списке разделяющих эту строку;  
- после получения подтверждений о том, что все копии заменены, резидентный узел пересылает разрешение продолжить вычисления в процессор, приостановленный после записи  
Недостатком является время простоя, DASH лучше.*

Алгоритм DASH. Каждый блок памяти имеет для каждой строки, резидентной в узле, список узлов, в кэшах которых размещены копии строк. Каждая резидентная строка может находиться в трех глобальных состояниях:

1. Некэшированнная – копия строки не находится в кэше узлов, кроме, возможно резидентного для этой строки
2. Удаленно-разделенная – копии строки размещены в кэшах других узлов.
3. Удаленно-измененная – строка изменена операцией записи в каком-либо узле

Также строка кэша может находиться в одном из 3 локальных состояний:

1. Невозможная к использованию
2. Разделяемая - есть неизменная копия, которая может размещаться в других кэшах
3. Измененная – если копия изменена операцией записи.

*Реализация алгоритма когерентности с большим быстродействием возможна за счет учета специфики параллельных про­ грамм, в которых асинхронно используются одни и те же переменные на каждом временном интервале только одним процессором с последующим переносом обработки на другой процессор. Такой тип вычислений позволяет организовать более эффективные схемы передачи необходимых строк из кэша одного процессора в кэш другого.*

## 6.3 Явная когерентность(программная).

Основная задача программиста- эффективное программирование параллельных процессов, совмещающих вычисления и передачу данных между узлами и минимизирующих объем передаваемой информации. Тк память в узлах является могоуровневой, то очевидно, что получаемые по коммуникационному интерфейсу строки данных делают несостоятельными копии этих строк в кэш памяти. Поэтому нужно предусмотреть организацию когерентности прибывшей и кэшированной строк.

Различают следующие варианты поддержки когерентности:  
- при наличии дубликатов признаков (тегов) строк кэш-памяти в контроллере прямого доступа делать по получении строки несостоятельной только действительно необходимую строку;  
- при отсутствии признаковой памяти в контроллере делать по каждому приему строки несостоятельными все строки кэш-памяти.  
Применение явной когерентности в создаваемых вычислительных системах обусловлено либо слишком большим временем, либо недопустимо высокими аппаратными затратами на реализацию неявного механизма когерентности.

## Механизм когерентности на основе масштабируемого когерентного интерфейса SCI.

SCI принял в качестве стандарта ANSI. Реализация предусматривает дополнительный уровень кэш-памяти, размещенный в узле вычислительной системы. Поиск данный производится при промахе на предыдущих уровнях памяти и если данные присутствуют, то они и идут в верхние уровни памяти. Если данные в узле модифицируются, то по окончании также идут в верхние уровни. Если в узле нет данных, срабатывает промах. Стандарт SCI строит выч. системы с быстродействием более 1гбит/с.

## Коммуникационная среда MYRINET

Среда Myrinet стандартизует формат пакета, способ адресации вычислительных машин (ВМ), протокол передачи пакетов. Коммуникационная среда образуется адаптерами ’’шина компьютера - линк сети” и коммутаторами линков сети. Каждый ЛИНК содержит две однонаправленных линии и образует канал с общей пропускной способностью равной 80 Мбайт/с. Согласно требованиям, сеть Myrinet не должна блокироваться из-за неисправностей или обесточенных устройств. Если линк подключен к неработоспособному адаптеру, то передача не блокируется, но передаваемые пакеты теряются.

## Средства распараллеливания для систем с общим полем памяти (стандарт ОреnМР).

До появления OpenMP программы автоматически распараллеливались компилятором, используя ручную (или же явную) организацию параллельных задач с помощью обращений к специальной библиотеке примитивов. Стандарт же OpenMP является промежуточным подходом к распараллеливанию программ с применением “директив пользователя” и распараллеливающего по подсказкам пользователя компилятора. “Директивы пользователя” указывают компилятору на параллелизм

## Средства распараллеливания для систем с распределенной памятью (стандарт MPI)

Интерфейс передачи сообщений MPI включает в себя понятия процесса, группы процессов и коммуникатора. Процесс-исполнение программы на одном процессоре. Коммутатор является средой, для взаимодействия процессоров и обеспечивает обмен данными между ними и синхронизацию. Также есть внутригрупповые и межгрупповые коммуникаторы.

# 7. Принципы оценки производительности вычислителей

7.1. Цели исследований и показатели производительности

**ЦЕЛИ:**

* оценка с целью выбора ВС среди альтернатив для приобретения
* оценка производительности разрабатываемой ВС или ее компонент (планирование производительности или проектирование с заданной производительностью)
* контроль производительности (сбор и накопление данных о характеристиках ВС для прогноза влияния планируемых изменений конфигурации и ОС на производительность ВС)

В общем случае для оценки производительности необходимо предсказать характер прикладных задач, решаемых на ВС, и нагрузку на ВС.

**ПОКАЗАТЕЛИ ПРОИЗВОДИТЕЛЬНОСТИ:**

1. цикл выполнения задания в пакетных системах (время от момента поступления задания в ВС до его выполнения и возвращения результатов пользователю);
2. время ответа для интерактивных систем (время от момента ввода данных с клавиатуры до момента начала ответа);
3. дисперсию времени ответа, являющуюся мерой предсказуемости времени ответа в диалоговых системах; - время реакции в системах реального времени (время от момента ввода данных до момента выделения кванта на обслуживание запроса);
4. пропускную способность (число обслуженных заданий, запросов, транзакций в единицу времени);
5. загрузку ресурсов ВС.

Для **оценки** данных показателей используются следующие методы измерений:

1. Измерение элементарных времен, позволяющее оценить техническую производительность компонент и узлов вычислителя;
2. Измерение времени выполнения смеси команд, характерной для применения данной вычислительной установки;
3. Время выполнения образцовой программы, типичной для применения данной ВС (метод часто используется при построении компиляторов для выбора наиболее эффективного кода языковых конструкций программы);
4. Время выполнения измерительных программ, типичных для решаемого на ВС класса задач (обычно данный метод применяется при намерении сменить ВС или ОС для эксплуатации готового программного обеспечения);
5. Время выполнения синтетических программ, совмещающих в себе черты образцовых и измерительных программ;
6. Аналитическое моделирование процессов, протекающих в ВС. Аналитические модели позволяют быстро и наиболее наглядно увидеть достоинства и недостатки ВС, однако формализация вычислительных процессов всегда предусматривает их идегилизацию, после чего возникает вопрос адекватности аналитической модели реальной системе и проблема интерпретации полученных результатов;
7. Имитационное моделирование ВС, позволяющее исследовать трудноформализуемые при аналитическом моделировании элементы вычислительного процесса.

7.2. Пиковая и реальная производительность

**Пиковая** производительность определяет теоретический максимум быстродействия вычислительной системы в идеальных условиях.

Пиковая производительность определяется числом операций, выполняемых всеми исполнительными устройствами вычислителя, и равна произведению тактовой частоты, числа арифметическо-логических устройств процессора и количества процессоров вычислителя.

Измеряется в миллионах операций в секунду MIPS или миллионах операций с плавающей точкой в секунду MFLOPS.

**Реальная** производительность определяется классом решаемых задач. Время выполнения каждой из задач - основа для расчета индекса производительности данной вычислительной системы.

7.3. Тесты производительности

Три проблемы, связанные с анализом результатов тестов производительности:

* проблема достоверности оценок,
* проблема адекватности оценок (выбора тестов, наиболее точно характеризующих производительность при обработке типовых задач),
* проблема интерпретации (правильного истолкования результатов тестирования).

**ГРУППЫ ТЕСТОВ:**

Тесты можно разбить на **три группы**:

**Тесты производителей** средств вычислительной техники, используемые для оценивания качества собственных про­дуктов. Например, индекс ICOMP, индекс оценивает фактически производительность микропроцессора, а не вычислительной установки.

Стандартные **тесты, создаваемые независимыми аналитиками** и используемые для сравнения широкого спектра компьютеров. Например, тестовый пакет Linpack, тестовые наборы SPEC XX, тестовые пакеты ТРС, тестовый комплекс WebStone.

**Пользовательские тесты**, учитывающие специфику конкретного применения вычислительной системы. Данные тесты создаются крупными компаниями и используются для выбора средств вычислительной техники и программного обеспечения под определенные прикладные задачи. К этому классу можно отнести комплекс тестов NAS для оценки производительности многопроцессорных суперкомпьютеров с МРР-архитектурой.

**ТЕСТЫ:**

Комплекс **тестов NAS** общепризнанно считается лучшим тестовым набором для оценки многопроцессорных вычислительных систем с массовым параллелизмом. Тесты позволяют оценить вычислительные возможности компьютерной системы и скорость передачи данных между процессорами в массово-параллельных системах. Метрикой теста является относительная производительность по сравнению с показателями традиционного векторного суперкомпьютера

**Тесты Linpack** представляют собой совокупность программ решения задач линейной алгебры. Параметрами тестов являются порядок матрицы, формат значений элементов матрицы, способ компиляции, возможность применения оптимизированной библиотеки стандартных функций.

**Пакет SPEC 89**. Методика оценки производительности SPEC89 предполагает формирование десяти оценок SPECratioi, каждая из которых определяется как отношение времени выполнения i-й программы на тестируемом компьютере ко времени выполнения той же программы на вычислителе DEC VAX 11/780. SPECmark является средним геометрическим всех десяти частных оценок SPECratio.

К параметру SPECmark добавляются еще две оценки - SPECint89 и SPECfp89, раздельно характеризующие быстродействие компьютера при обработке целочисленных данных и вещественных чисел.

**Пакет тестовых программ SPEC 92** расширяет набор тестируемых функций по сравнению со SPEC89.

Также существуют **тестовые программы SPEC 95**.

Комплект процессорных **тестов SPEC2000** предназначен для тестирования процессора, иерархической памяти, компиляторов

**Пакет тестовых программ ТРС** применяется для оценки производительности при работе с базами данных.

Тесты ТРС включают набор тестов для измерения эффективности функционирования в различных режимах обработки данных: ТРС-А, ТРС-В, ТРС-С, TPC-D, TPC-W.

В основе тестового набора ТРС лежат следующие фундаментальные принципы:

* производительность соотносится с общей стоимостью тестируемой системы, включая аппаратную составляющую, программное обеспечение и эксплуатационные издержки в расчете на заданный срок эксплуатации;
* тест формулируется на высоком функциональном уровне без связи с конкретной программно-аппаратной платформой, допуская тем самым сравнение между собой различных реализаций;
* тест должен учитывать потенциальный рост мощности вычислительных систем и допускать масштабирование нагрузки по числу пользователей и объему базы данных.

*Тест ТРС-А используется для оценки эффективности исполнения транзакций типа ’’дебет-кредит”.*

*Тест ТРС-В аналогичен тесту ТРС-А, но ориентирован на пакетный режим обработки и имеет другую методику подсчета стоимости ВС.*

*Тест ТРС-С является основным продуктом ТРС и предназначен для оценки эффективности ВС при работе в режиме оперативной обработки транзакций*

*Тест TPC-D используется для оценки эффективности сложных информационных систем поддержки принятия решений с многонаправленными соединениями, сортировками и агрегированием данных отношений.*

*Тест TPC-W предназначен для оценки систем поддержки электронной коммерции, электронного бизнеса, коммерческих Web-приложений, корпоративных сетей.*

**Тест WebStone** используется для оценки конфигураций Web. Тест WebStone позволяет дать объективную оценку аппаратуры, программного обеспечения и дисциплины взаимодействия с сетью. Данный тест отражает специфику работы в глобальной сети с многократными переключениями, коррекцией ошибок, переадресациями и т.д. Основными метриками теста WebStone являются пропускная способность и время выполнения запроса (задержка), которые усредняются по множеству измерений за сеанс тестирования.

3 файл

## Уровни RAID

На уровне дисковой подсистемы предпринимаются технические меры, направленные на снижение потерь до стоимости вышедшего из строя оборудования. В основе таких мер лежит резервирование и избыточность, реализуемые в виде массива независимых резервных дисков —**RAID.**

Всего выделяют 9 уровней RAID-массивов, различающихся по скорости, надежности и стоимости изготовления.

**RAID 0** — дисковый массив без дополнительной отказоустойчивости

**RAID 1** – дисковый массив с зеркалированием данных

**RAID 2** — дисковый массив с использованием алгоритма Хэмминга для проверки/восстановления данных

**RAID 3** — дисковый массив с вычислением контрольной суммы параллельно с передачей данных

**RAID 4** — дисковый массив с независимыми дисками данных и общим диском для хранения контрольных сумм

**RAID 5** — дисковый массив с независимыми дисками данных и равномерным распределением контрольных сумм между дисками

**RAID 6** — дисковый массив с независимыми дисками данных и двумя независимыми схемами контрольных сумм, распределенными между дисками

**RAID 7** — дисковый массив с асинхронным вводом-выводом и высокой скоростью передачи данных

**RAID 10** — комбинация технологий RAID 1 и RAID 0 в одном дисковом массиве

**RAID 53** — комбинация технологий RAID 0 и RAID З в одном дисковом массиве

## Как понять, что объект в кэше?

Информационная память кэша и оперативная память вычислительной системы разбиты на блоки одинаковой длины и каждый блок памяти отображается на одну из строк кэша. В большинстве систем размер блока - 32 или 64 байта. Адрес объекта в памяти определяет строку кэша, в которую помещается блок информации, содержащий адресуемый объект. При обращении центрального процессора к адресуемому объекту выполняются следующие действия:

1. Из адреса объекта выделяются три поля:

— **номер байта в блоке**, занимающий младшие разряды адреса объекта;

— **индекс, определяющий адрес блока в кэше,** в котором должен находиться адресуемый объект, и занимает средние разряды адреса объекта;

— **признак,** занимающий старшие разряды адреса и позволяющий отличить блоки оперативки; при записи блока памяти в строку КЭШа, определяемую индексом, в элемент теговой памяти строки помещается признак.

Длина блока, количество блоков в кэше, емкость кэша и объем кэшируемой памяти - степень двойки (поля адреса занимают целое число разрядов).

1. По индексу из адреса объекта находится строка кэша, в которой находится нужный блок памяти.
2. Признак из адреса объекта, сравнивается с признаком из элемента признаковой памяти найденной строки кэша. Если признаки совпали, то адресуемый объект в кэше — он считывается из блока строки, на которую указывает индекс, и передается в процессор. Если же нет, то нужная информация в кэше отсутствует и в выбранную строку считывается блок из основной памяти.

Если длина элемента данных превышает один байт, то возможны ситуации, когда этот элемент (частями) расположен в двух (или более) разных КЭШ-строках, тогда время на выборку такого элемента увеличится. Противодействовать этому можно, выравнивая операнды и команды по границам КЭШ-строк.

## От чего зависит быстродействие ячейки оперативной памяти

От скорости изменения заряда конденсатора.

## От чего зависит пропускная способность шины

Измеряется в килобайтах (мегабайтах) в секунду и определяется ее тактовой частотой и разрядностью (шириной) шины данных.

## Ложные и истинные взаимозависимости

**Истинная взаимозависимость данных** или взаимозависимость “чтение после записи” **-**входные данные для одной операции зависят от результата другой, (например – a\*b+c). **Анти-взаимозависимость**, или взаимозависимость “ запись после чтения“ возникает, когда вторая команда может испортить данные, необходимые в качестве входных для первой. **Переименование регистров** – нейтрализация ложных взаимозависимостей путем переименования регистров. RISC-процессоры имеют достаточное кол-во регистров чтобы ложные взаимозависимости не появлялись часто, в то время как CISC-процессоры имеют **небольшое кол-во архитектурных регистров, что провоцирует частое возникновение ложных взаимозависимостей**. Если команде требуется регистр, процессор динамически ставит в соответствие этому логическому регистру один из множества физических регистров. Если другая команда пытается обратиться к тому же логическому регистру, то процессор для предотвращения конфликта может поставить ему в соответствие другой физический регистр. Такие переименования действуют, пока команды продвигаются по конвейерам. Только переименование регистров не решает проблемы ложной взаимозависимости, поэтому используется **обход данных –** результаты выполнения одной команды сразу пересылаются следующей и таким образом исключаются задержки на модификацию и повторное чтение из регистра или памяти. Так же используется **продвижение данных** - процессор может выполнять некоторые команды параллельно, немедленно передавая результаты одной из них в другую, которой они потребуются на более поздней ступени конвейерной обработки. При этом также исключается ожидание завершения операций записи и чтения адресуемых операндов.

## Соотношение CISC и RISC архитектур

**CISC** - архитектура вычислений с полной системой команд. Выполнение команд обработки данных, сохраняемых в памяти (операции "память­-память").

**RISC** - архитектура вычислений с сокращенным набором команд. Подразумевает обработку данных, хранящихся только в регистрах. **Обе архитектуры являются воплощением SISD-архитектур.** RISC- архитектура производительнее в тех приложениях, где над каждой единицей данных выполняется большой объем вычислительной работы (инженерные и научные задачи), в противном случае часто приходится загружать данные из памяти.

## Фазы классического конвейера

В большинстве классических СISC-архитектур имеется пять ступеней обработки конвейера:

1) Выборка команды

2) Декодирование команды

3) Адресация и выборка операндов

4) Исполнение команды

5) Запись результата

## Для чего нужно переименование регистров

**Переименование регистров** – нейтрализация ложных взаимозависимостей путем переименования регистров. RISC-процессоры имеют достаточное кол-во регистров чтобы ложные взаимозависимости не появлялись часто, в то время как CISC-процессоры имеют **небольшое кол-во архитектурных регистров, что провоцирует частое возникновение ложных взаимозависимостей**. Если команде требуется регистр, процессор динамически ставит в соответствие этому логическому регистру один из множества физических регистров. Если другая команда пытается обратиться к тому же логическому регистру, то процессор для предотвращения конфликта может поставить ему в соответствие другой физический регистр. Такие переименования действуют, пока команды продвигаются по конвейерам. Только переименование регистров не решает проблемы ложной взаимозависимости, поэтому используется **обход данных –** результаты выполнения одной команды сразу пересылаются следующей и таким образом исключаются задержки на модификацию и повторное чтение из регистра или памяти. Так же используется **продвижение данных** - процессор может выполнять некоторые команды параллельно, немедленно передавая результаты одной из них в другую, которой они потребуются на более поздней ступени конвейерной обработки. При этом также исключается ожидание завершения операций записи и чтения адресуемых операндов.

## UMA от NUMA отличия

**СС-NUМА** -системы с кэш когерентным доступом (при изменении объекта одним процессором или ядром, изменяется объект для всей системы) к неоднородной памяти. Системы этого класса строятся из узлов с SМР­ архитектурой, имеющих собственную (локальную) область общего адресного пространства и кэш "дальней" памяти (областей памяти других узлов). Узлы между собой связываются с помощью шины. По существу, данная архитектура является расширением систем с симметричной мультипроцессорностью. В отличие от классической архитектуры NUMA, при использовании кэш-когерентного доступа к неоднородной памяти все процессоры объединены в один узел, причем первый уровень иерархии памяти образует кэш-память процессоров.

**UMA** - системы с однородной памятью. Имеют общую (разделяемую) память для всех процессоров вычислителя. Архитектура предполагает наличие одной (реже двух или более) общей шины к которой подключены несколько процессоров. Доступ к любому элементу данных происходит единообразно.

**NUMA** –системас неоднородным доступом к памя­ти. Наличие индивидуальной оперативной памяти возле каждого процессора (или группы процессоров) и поддержка неоднородного доступа к памяти, обеспечивающая существенно различное время обращения к объектам в распределенной памяти. Каждый процессор имеет доступ не только к собственной ло­кальной памяти, но и к памяти других процессоров сети. Но поскольку при обращении к памяти других процессоров запросы проходят через сеть, они вы­полняются дольше, чем обращения к локальной памяти. Системы этого типа называются мультипроцессорными системами. В системе CC-NUMA **физически распределенная память объединяется в единый массив** (единое адресное пространство с частями, физически связанными специальной соединительной шиной, и аппаратно-реализованная кэш-когерентность).

**С точки зрения модели программирования:**

-Ключевое различие – диктуемая модель программирования. Задержки доступа к адресуемым объектам происходят из-за различия в способах программирования.

· SMP-, СМР-и СС-МЈМА-системы попадают в один класс компьютерных систем с общей памятью и аппаратно-поддерживаемой когерентностью данных. (+ доступ к часто используемым данным, в разделяемой памяти, за микросекунды, в то время как считывание их с диска требует миллисекунд)

· МРР-системы и вычислительные кластеры — в класс вычислительных систем с распределенной памятью и программно-реализуемой когерентностью данных (-время доступа к объектам в удаленной памяти повышается до сотни микросекунд — это в сотни раз медленнее, чем быстродействие локальной памяти)

**Наиболее выгодно программирование в SMP**, поскольку при этом **программисту не нужно заботиться о распределении данных в памяти**, так как все ее части доступны для любого процессора и доступ к ним одинаково быстр.

## Что будет происходить при увеличении длины кэшируемой строки

Разрядность индекса уменьшится, размер поля увеличится, размер признака увеличится.

## Отличие SMP от MPP

**SМР-системы** - системы мультипроцессорной обработки с разделяемой памятью. Вычислительные системы с сильной связью, называемые также системами с симметричной мультипроцессорностью, ­ **основаны на объединении процессоров на общем поле памяти**, которая совместно с интерфейсной шиной или коммутатором "процессоры-память" при увеличении числа процессоров становится узким местом системы. **Основное достоинство SМР-систем** заключается в простой модели программирования приложений (разработчику доступны в общем поле памяти все обрабатываемые переменные).

**МРР­ системы** - системы с массовым параллелизмом. Вычислительные системы со слабой связью. Состоят из узлов (процессор, память, система ввода ­ вывода), работающих под управлением своей копии операционной системы на своем уникальном адресном пространстве. **Основная проблема вычислителей данного класса** состоит в сложности модели программирования, предполагающей распределение обрабатываемых данных по различным узлам и явную программную реализацию когерентности с помощью техники передачи сообщений. Для хранения обрабатываемых данных используют распределённую по узлам вычислителя память с индивидуальным в каждом узле адресным пространством.

## Представители вычислителей класса MIMD

**МIМD - архитектура** включает все уровни параллелизма от конвейера операций до независимых заданий и программ. Любая вычислительная система этого класса в частных приложениях может выступать как SISD (одиночный поток команд и данных) или SIMD (одиночный поток команд и множество потоков данных) - система.

## Прием неупорядоченной обработки команд

Позволяет процессору “ выдавать" и “ обрабатывать“ команды также не в порядке их следования по программе. Для реализации данной обработки требуется **буфер команд** между ступенями декодирования и исполнения конвейера. Этот прием используется для повышения производительности за счет увеличения числа исполняемых команд за такт.

## Превращение CISC архитектуры в RISC

Основа успеха микропроцессора на массовом рынке — сохранение совместимости с существующими и будущими приложениями. Для x86 это трудная задачи из-за сложной системы команд, особенностей регистров системного уровня и механизмов защиты памяти и механизмов обработки внешних событий. Решением стало преобразованием x86 команд во внутренние RISC команды.

Главная проблема декодирования CISC команд заключается в определении границ команд и в определении наиболее эффективной последовательности выполнения дешифратором. AMD решили проблему путем распознания сложных команд до стадии декодирования. После определения сложных команд создаются биты предварительного декодирования. Каждый байт в кэш-памяти имеет биты данных декодирования с адресами соответствующих RISC-подобных операций. Процесс предварительного декодирования, происходящий при выборке команд из оперативной памяти в кэш команд, практически не снижает производительности системы, но требует дополнительной кэш-памяти.

## Зачем нужно прогнозирование?

Предсказание ветвлений позволяет сократить время простоя конвейера за счёт предварительной загрузки и исполнения инструкций, которые должны выполниться после выполнения инструкции условного перехода. Без предсказания переходов конвейер должен дождаться выполнения инструкции условного перехода, чтобы произвести следующую выборку. Предсказатель переходов позволяет избежать траты времени, пытаясь выяснить ответвление. Ответвление выбирается по предыдущим результатам проверки условия. Предполагаемое ответвление затем загружается и частично выполняется. Если затем обнаруживается, что предсказание было выполнено неверно, отменяются результаты неверного ветвления и в конвейер загружается правильное ответвление, производя задержку. Существует 2 метода прогнозирования:

* **Статические методы** предписывают всегда выполнять или не выполнять определенные типы переходов.
* **Динамические методы** оценивают поведение команд перехода за предшествующий период времени.

В Р6 используется исполнение по предположению — исполнение команд, следующих за командой условного перехода до того, как выяснится, правильно ли был предсказан переход. При этом процессор не обновляет архитектурные регистры и память до однозначного разрешения команд, выполненных по предположению. При неправильном предсказании ROB процессора Р6 отбрасывает исполненные по предположению команды, прежде чем они будут удалены.

Как происходит:

Устраняет процедурные зависимости, то есть условные и безусловные переходы. Увеличивает эффективность суперскалярности, неупорядоченной обработки и продвижения данных. Процессор использует “предложение о пути ветвления” и после этого начинает: 1. Выборку команд с предсказанного адреса перехода до того, как он узнает, верным ли был его прогноз. 2. Исполнение по предположению “спекулятивное”, при этом процессор не может модифицировать архитектурные регистры или память до однозначного разрешения перехода. Обычно используется буфер адреса перехода. Бывает предикатное выполнение команд: выполнение по всем возможным исходам условного перехода.

## Что делать для того, чтобы снизить время выполнения фаз конвейера?

Кроме того, производительность может понижаться из-за:

1) Взаимозависимость по выходу или “запись после записи” возникает при выполнении двух, следующим друг за другом команд, которые записываются в 1 и тот же регистр. Процессор при этом должен гарантировать правильность модификации регистра, даже если команды, модифицирующие его, выполняются не по порядку;

2) Анти-взаимозависимость, или взаимозависимость “ запись после чтения“ возникает, когда вторая команда может испортить данные, необходимые в качестве входных для первой.

**Факторы, нарушающие непрерывность конвейерной реализации команд:** 1) Когда для выполнения следующей команды требуется результат действий, реализованных предыдущей командой, или, когда предыдущей командой определяется операнд следующей команды, следовательно, возникает задержка выполнения след. команды.

2) когда в кэш-памяти отсутствуют требуемые данные или команды, необходимо передать их в кэш-память из основной памяти;

3) когда предшествующая команда изменяет содержание следующей или когда изменяется содержимое регистра состояния программы, задающего область выполнения программы, следующая команда должна ожидать завершения предшествующей команды;

4) в случае возникновения прерывания и перехода к программе его обработки (конвейер загружается командами обработки прерывания)

5) когда операция, реализуемая машинной командой, имеет сложный характер и требует для выполнения много машинных циклов, последующая команда долго не может достичь стадии выполнения операции.

## Прием для быстрого выбора команд

Кэширование и ускорение работы памяти, которая в конвейерном виде позволяет считывать байты одного слова.

## Какие фазы конвейера обходятся при его обходе

Переименование данных не может полностью исключить возникновение истинных взаимозависимостей, потому используются 2 метода:

1) **ОБХОД ДАННЫХ** – При обходах результаты выполнения одной команды сразу пересылаются следующей и таким образом исключаются задержки на модификацию и повторное чтение из регистра или памяти.

2) **ПРОДВИЖЕНИЕ ДАННЫХ** - при продвижении данных процессор может выполнять некоторые команды параллельно, немедленно передавая результаты одной из них в другую, которой они потребуются на более поздней ступени конвейерной обработки. При этом также исключается ожидание завершения операций записи и чтения адресуемых операндов

## Для чего нужна векторная обработка?

Для обработки циклов и итераций.

## Как загрузить объект в кэш

С помощью методов сквозной и обратной записи.

1. **Метод сквозной записи** - информация записывается как в кэш, так и в оперативку. Однако это снижает производительность системы, т.к. при каждой записи необходимо тратить время на запись в оперативную память. 2. **Метод обратной записи** - сохраняет блок кэша в оперативке только при вытеснении его из кэша и при наличии в нем изменений. Для этого каждому блоку кэша ставится бит изменений, который устанавливается в случае записи новых данных. При замене старого блока информации кэша на новый из оперативки проверяется состояние соответствующею бита.

## Что такое кэш с обратной записью

Метод обратной записи позволяет сохранять блок кэша в оперативной памяти только при вытеснении его из кэша и при наличии в нем изменений. Для этого каждому блоку кэша ставится в соответствие бит изменений, который устанавливается в том случае, если в блок были записаны новые данные. При замещении старого блока информации в кэше на новый из оперативной памяти проверяется состояние соответствующего бита. Если он установлен, то сначала выполняется перепись блока из кэша в память, и только после этого в кэш помешается новый блок из памяти. Такой метод обеспечивает более высокую производительность вычислительной системы, так как количество измененных блоков обычно меньше числа операций записи в память.

## Сколько в группе строк кэша

Столько же, чему равен коэффициент ассоциативности(А). А = 1 множественный ассоциативный кэш превращается в кэш прямого отображения, а при А, равном числу строк в кэше, - в полностью ассоциативный кэш.

## На уровне какого параллелизма используется многофункциональная обработка

На уровне операторов и команд

## Имеется кэш, емкость которого увеличилась, что изменилось

Увеличилось кол-во строк.

## Увеличилась ассоциативность в 2 раза что изменилось?

С ростом ассоциативности кэша разрядность признакового поля растет, а разрядность индекса падает.

## Сквозная запись

Метод сквозной записи заключается в том, что информация записывается как в кэш, так и в оперативную память. Однако такое техническое решение снижает производительность системы, поскольку при выполнении каждой операции записи необходимо тратить время на запись в оперативную память.

## Что произойдет с кэшем если увеличится объем кэшируемой памяти

С ростом длины блока при постоянной емкости кэша количество строк кэша уменьшается, и разрядность индекса также снижается, размер поля ’’номер байта в блоке” растет, а объем признакового поля не изменяется.

## Из чего состоит параллельная шина?

По способу передачи данных шины делятся на последовательные и параллельные. Последовательные шины передают данные по одному проводнику, один бит за один раз, в параллельных шинах передача данных разделена между несколькими проводниками и поэтому можно передать большее количество данных.

## Что такое VLIW?

**VLIW** – архитектура с очень длинным командным словом. На этапе компиляции несвязанные операции группируются в пакеты, содержимое которых строго соответствует структуре процессора. Таким образом, в рамках данной архитектуры компилятор находит в исходном коде инструкции, выполнять которые можно параллельно, и создает машинный код, использующий этот параллелизм.

**Скалярный процессор** – процессор с единственным конвейером. **Суперскалярный процессор** – процессор с несколькими конвейерами. Двух потоковые – dual-issue. Четырех потоковые – quad-issue.

## Кластеры

Кластеры – разновидность MPP-систем.

Кластер состоит из двух или более узлов, удовлетворяющих следующим требованиям:

1. Каждый узел работает со своей копией ОС.

2. Каждый узел работает со своей копией приложения.

3. Узлы делят общий пул других ресурсов (в кластерах приложений). Отдельные экземпляры кластера должны знать о работе друг друга.

## Принципы выбора архитектуры.

Из всех возможных вариантов построения системы лучшим является тот, который обеспечивается наиболее простой архитектурой.

1. Производительность компьютера определяется главным образом двумя характеристиками – тактовой частотой и разрядностью обработки. Конвейерная организация процессора “работает” на усиление значимости тактовой частоты и является необходимым атрибутом современных компьютеров.

2. Для высокопроизводительных вычислений целесообразно применять RISC-процессоры или процессоры с RISC-ядром. Системы на их основе специально ориентированы на повышение тактовой частоты обработки и “прозрачность” параллелизма архитектуры для прикладного программиста. 3. Предпочтительней использовать один быстрый однопроцессорный компьютер, чем много медленных или многопроцессорную систему с маломощными процессорами.

4. Проблему создания необходимого количества рабочих мест лучше решать с помощью многопользовательских систем на базе сервера и терминалов, чем с помощью ЛВС персональных компьютеров.

5. Вычислительные системы с массовым параллелизмом следует использовать только при полной уверенности в реально существующем параллелизме приложений и только при возможности привлечения программистов высшей квалификации.

## Блок вычислений с плавающей точкой

Операции с плавающей точкой имеют высокую трудоемкость, поэтому блоки вычислений с плавающей точкой реализуются в виде многоступенчатых конвейеров. SIMD-реализации обработки векторных данных.

## Кэш память микропроцессора

Реализация на кристалле раздельного кэша данных и команд. С ростом размера кристалла растет число брака.

4 файл

Архитектура вычислительных систем.

Лекция Х. Диски – внешняя энергонезависимая память (энергонезависимое запоминающее устройство, которое использует флэш-память для хранения информации).

Сравнительный анализ организации дисковых устройств.

Основные характеристики твердотельных накопителей[[14]:](https://ru.wikipedia.org/wiki/%D0%A2%D0%B2%D0%B5%D1%80%D0%B4%D0%BE%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BD%D0%B0%D0%BA%D0%BE%D0%BF%D0%B8%D1%82%D0%B5%D0%BB%D1%8C#cite_note-_8631e98a89666d1a-14)

* наименьшее время доступа к данным: от 100 до 1000 раз быстрее, чем у механических дисков;  высокая скорость, до нескольких гигабайт в секунду для произвольно расположенных данных;
* высокие значения IOPS благодаря высокой скорости и низкому времени доступа;  низкая цена производительности, лучшее соотношение цены к производительности среди всех устройств хранения;
* высокая надёжность, SSD дают уровень сохранности данных такой же, как другие полупроводниковые устройства.

В отличие от жестких дисков, цена SSD очень сильно зависит от доступной емкости, это связано с ограниченной плотностью размещения ячеек памяти и ограничением размера кристалла в микросхеме[[15].](https://ru.wikipedia.org/wiki/%D0%A2%D0%B2%D0%B5%D1%80%D0%B4%D0%BE%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BD%D0%B0%D0%BA%D0%BE%D0%BF%D0%B8%D1%82%D0%B5%D0%BB%D1%8C#cite_note-%D0%9E%D0%B1%D0%B5%D1%80,2016,3-15)

Преимущество SSD дисков по сравнению с традиционными накопителями на жёстких дисках:

* + высокая механическая надёжность, – отсутствие движущихся частей,
  + высокая скорость чтения/записи,
  + низкий вес,
  + меньшее энергопотребление.

Но всё ли так хорошо?

## SSD накопитель

SSD – это твердотельный накопитель (*SSD, Solid State Drive или Solid State Disk*), энергонезависимое, перезаписываемое запоминающее устройство без движущихся механических частей с использованием флэш-памяти. SSD полностью эмулирует работу жёсткого диска.

SSD – это большая флэшка. Но в отличие от флэшек, в SSD используется микросхема DDR DRAM (Double Data Rate Dynamic Random

Access Memory) кеш-памяти, в связи со спецификой работы и возросшей в несколько раз скоростью обмена данными между контроллером и интерфейсом SATA.

## Устройство контроллера SSD диска

Главной задачей контроллера является обеспечение операций чтения/записи, и управление структурой размещения данных. Основываясь на матрице размещения блоков, в какие ячейки уже проводилась запись, а в какие еще нет, контроллер должен:

* оптимизировать скорость записи,
* обеспечить максимально длительный срок службы SSD-диска.

Вследствие особенностей построения NAND-памяти, работать с ее каждой ячейкой отдельно нельзя. Ячейки объединены в страницы объемом по 4 Кбайта, и записать информацию можно только полностью заняв страницу. Стирать данные можно по блокам, которые равны 512 Кбайт. Эти ограничения накладывают определенные требования к интеллектуальному алгоритму работы контроллера. Правильно настроенные и оптимизированные алгоритмы контролера могут существенно повысить производительность и долговечность работы SSD-диска.

В контроллер входят следующие основные элементы:

* **Processor** – как правило 16 или 32 разрядный микроконтроллер. Выполняет инструкции микропрограммы, отвечает за перемешивание и выравнивание данных на Flash, диагностику SMART, кеширование, безопасность.
* **Error Correction (ECC)** – блок контроля и коррекции ошибок ECC.
* **Flash Controller** – включает адресацию, шину данных и контроль управления микросхемами Flash памяти.
* **DRAM Controller** - адресация, шина данных и управление DDR/DDR2/SDRAM кэш памятью.
* **I/O interface** – отвечает за интерфейс передачи данных на внешние интерфейсы SATA, USB или SAS.
* **Controller Memory** – состоит из ROM памяти и буфера.

Память используется процессором для выполнения микропрограммы и как буфер для временного хранения данных. При отсутствии внешней микросхемы RAM памяти выступает в роли единственного буфера данных SSD. Контроллер является слабым местом твердотельных SSD дисков, так как он более чувствителен к проблемам с питанием и повреждение микропрограммы (прошивки), находящейся в нем, может привести к полной потере всех данных пользователя. А их корректное восстановление — еще более трудозатратная операция, чем в случае с HDD дисками. В силу того, что данные разбросаны по разным чипам памяти и необходимо корректно восстановить первоначальную их структуру, а это бывает не просто.

# Основные элементы SSD:

* **PCB** — печатная плата.
* **NAND-flash** — флэш-память NAND; отвечает за хранение данных.
* **NAND-controller** — контроллер памяти; выступает в роли посредника между носителем и системой, и является процессором, отвечающим за производительность SSD.
* **DRAM** — кэш (присутствует не во всех моделях SSD); выступает временным хранилищем небольшого объема данных и позволяет стабилизировать износ памяти, а также ускорить доступ к файлам.
* **HOST Interface** — интерфейс подключения; тип соединения и протокол, через которые SSD соединяется с вашей системой.

## Flash память

В SSD как и в USB Flash используются три типа памяти NAND:

* SLC (Single Level Cell), – MLC (Multi Level Cell),
* TLC (Three Level Cell).

(С 2017 года начато использование QLC (Quad Level Cell) — четыре бита).

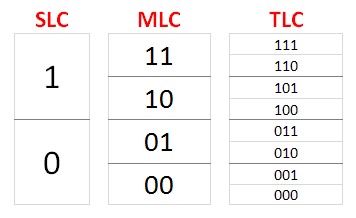
Отличие состоит в том, что SLC позволяет хранить в каждой ячейке только один бит информации, MLC – два, а TLC – три ячейки (использование разных уровней электрического заряда на плавающем затворе транзистора), что делает память **MLC и TLC** более дешёвой относительно ёмкости.

NAND-флэш является основным и самым дорогостоящим компонентом твердотельного накопителя. Выделим три основных типа памяти, используемых в современных SSD:

* **Planar NAND / 2D NAND** — устаревший вариант NAND памяти, который характеризуется однослойной структурой расположения ячеек памяти.
* **3D NAND / V-NAND** — современный и самый распространенный на сегодняшний день тип флэш-памяти в твердотельных накопителях, который характеризуется многослойной структурой расположения ячеек памяти.
* **3D XPOINT** — совместная разработка Intel и Micron, являющаяся более быстрой и более дорогой альтернативой 3D NAND памяти.

Тип 3D NAND памяти, используемый в накопителе, напрямую влияет на его эффективность и долговечность. На данный момент в твердотельных накопителях вы можете встретить следующие типы 3D NAND памяти:

* **SLC** — имеет ячейки памяти, содержащие один бит информации; самая эффективная, долговечная и дорогая 3D NAND памятью, использующаяся в серверных и профессиональных накопителях.
* **MLC** — имеет ячейки памяти, содержащие два бита информации; следующая по характеристикам и цене после SLC.
* **TLC** — имеет ячейки памяти, содержащие три бита информации; самая распространенная память в потребительских устройствах массового сегмента, обеспечивающая хорошее соотношение цены и характеристик.
* **QLC** — имеет ячейки памяти, содержащие 4 бита информации, и является самой дешёвой и самой слабой по характеристикам памятью, при этом позволяя создавать на своей основе накопители большого объёма.



Самыми дорогими и надежными решениями являются устройства на SLC чипах. Почему? Они позволяют в каждой ячейке памяти хранить только один бит информации. В отличие он них, MLC и TLC чипы могут хранить два и три бита соответственно. Это стало возможным за счет использования разных уровней электрического заряда на затворах ячеек памяти. С каждым новым уровнем

* + усложняется задача безошибочного распознавания уровня электрического сигнала,
  + а значит — увеличивается общее время поиска нужной ячейки с данными,
  + повышается вероятность возникновения ошибок чтения. **Кроме того,** память **MLC/TLC** обладает:
  + **меньшим ресурсом** (100 000 циклов стирания у SLC, в среднем 10

000 для MLC, а для TLC до 5 000), – **худшим быстродействием**.

С каждым дополнительным уровнем усложняется задача распознавания уровня сигнала, увеличивается время поиска адреса ячейки, повышается вероятность ошибок. **Так как SLC-чипы намного дороже и объем их ниже, то для массовых решений применяют в основном MLC/TLC-чипы.** На данный момент MLC/TLC память активно развивается и по скоростным характеристикам приближается к SLC.

**Низкую скорость MLC/TLC** производители SSD накопителей **компенсируют алгоритмами чередования блоков данных между микросхемами памяти** (одновременная запись/чтение в две микросхемы флэш-памяти, по байту в каждую) по аналогии с RAID 0, а **низкий ресурс циклов чт/зп** - **перемешиванием и слежением за равномерным использованием ячеек**.

Плюс к этому в SSD резервируется часть объёма памяти (до 20%). Это недоступная память для стандартных операций записи/чтения. Она резервируется на случай износа ячеек, по аналогии с магнитными накопителями HDD, который имеет резерв для замены bad-блоков. Дополнительный резерв ячеек используется динамически, и по мере физического изнашивания основных ячеек предоставляется резервная ячейка на заменую

**Как работает SSD накопитель**

Для чтения блока данных в винчестере (HDD) сначала нужно вычислить, где он находится, потом переместить блок магнитных головок на нужную дорожку, подождать пока нужный сектор окажется под головкой чт/зп и произвести считывание. Хаотические запросы к разным областям жесткого диска сильно сказываются на времени доступа. При таких запросах HDD вынуждены постоянно «гонять» головки по всей поверхности «блинов» и даже переупорядочивание очереди команд спасает не всегда.

В SSD проще — вычисляем адрес нужного блока и сразу же получаем к нему доступ (прямой) на чтение/запись. Никаких механических операций — всё время уходит на трансляцию адреса и передачу блока. Чем быстрее флэш-память, контроллер и внешний интерфейс, тем быстрее доступ к данным.

НО при изменении/стирании данных в SSD накопителе все не просто. Микросхемы NAND флэш-памяти оптимизированы для секторного выполнения операций.

**Флеш-память пишется блоками по 4 Кб, а стирается по 512 Кб.** При модификации нескольких байт внутри некоторого блока контроллер выполняет следующую последовательность действий:

1. Считывает блок, содержащий модифицируемый блок во внутренний буфер/кеш;
2. Модифицирует необходимые байты;
3. Выполняет стирание блока в микросхеме флэш-памяти;
4. Вычисляет новое местоположение блока в соответствии с требованиями алгоритма перемешивания;
5. Записывает блок на новое место.

НО как только информация записана, она не может быть перезаписана до тех пор, пока не будет очищена. Проблема в том, что минимальный размер записываемой информации не может быть меньше 4 Кб, а стереть данные можно минимум блоками по 512 Кб. Для этого контроллер группирует и переносит данные для освобождения целого блока.

Вот тут и сказывается оптимизация ОС для работы с HDD. При удалении файлов операционная система не производит физическую очистку секторов на диске, а только помечает файлы как удаленные, и знает, что занятое ими место можно заново использовать. Работе самого накопителя (и его интерфейса) это никак не мешает.

Такой метод удаления помогает повысить производительность при работе с HDD, в то время как при использовании SSD становится проблемой. В SSD, как и в традиционных жестких дисках, данные все еще хранятся на диске после того, как они были удалены операционной системой. НО твердотельный накопитель SSD не знает, какие из хранящихся данных являются полезными, а какие уже не нужны и вынужден все занятые блоки обрабатывать по длинному алгоритму: – прочитать,

* модифицировать,
* снова записать на место, после очистки затронутых операцией ячеек памяти, которые с точки зрения ОС уже удалены.

**Следовательно**, **чем** больше блоков на SSD содержит полезные данные, **тем** чаще приходится прибегать к процедуре **чтение>модификация>очистка>запись**, вместо прямой записи. Здесь пользователи SSD сталкиваются с тем, что быстродействие диска заметно снижается по мере их заполнения файлами. Накопителю просто не хватает заранее стёртых блоков. **Максимум производительности демонстрируют чистые накопители, а вот в ходе их эксплуатации реальная скорость понемногу начинает снижаться.**

Ранее в интерфейсе ATA не было команд для физической очистки блоков данных после удаления файлов на уровне ОС. Для HDD они просто не требовались. Появление SSD заставило пересмотреть отношение к данному вопросу. В результате спецификация ATA пополнена новой командой **DATA SET MANAGEMENT** (более известной как **Trim)**. Она позволяет OC на уровне драйвера собирать сведения об удаленных файлах и передавать их контроллеру накопителя.

В периоды простоя, SSD самостоятельно осуществляет очистку и дефрагментацию блоков отмеченных как удаленные в ОС. Контроллер перемещает данные так, чтобы получить больше предварительно стертых ячеек памяти, освобождая место для последующей записи. Это позволяет сократить задержки, возникающие в ходе работы.

Но для реализации **Trim** необходима поддержка этой команды прошивкой накопителя и установленным в ОС драйвером. На данный момент только самые последние модели SSD «понимают» TRIM, а для старых накопителей нужно прошить контроллер для включения поддержки этой команды. Среди операционных систем команду Trim поддерживают: Windows 7, Windows Server 2008 R2, Linux 2.6.33, FreeBSD 9.0. Для остальных ОС необходимо инсталлировать дополнительные драйвера и утилиты.

Например, для SSD от Intel существует специальная утилита **SSD Toolbox**, которая может выполнять процедуру синхронизации с ОС по расписанию. Кроме оптимизации, утилита позволяет выполнять диагностику SSD и просматривать SMART-данные всех накопителей компьютера. С помощью SMART, можно оценить текущую степень износа SSD – **параметр E9 отражает оставшееся количество циклов очистки NAND-ячеек в процентах от нормативного значения**. Когда величина, уменьшаясь от 100, дойдет до 1, можно ожидать скорое появление «битых» блоков.

**Процедура дефрагментации** не только не нужна данному типу устройств (скорость доступа у них одинакова для любой ячейки не зависимо от того, где находится конечный файл), **но и вредна**. Лишние (холостые) обращения к диску только дополнительно снижают его ограниченный ресурс чт/зп. Значит — служба дефрагментации не нужна!

Фирма «Intel» утверждает, что идеальными условиями работы SSD твердотельного диска является его заполненность данными меньше чем на 75% с соотношением статической (редко изменяемой) и динамической (изменяемой часто) информации — 3 к 1. Не следует использовать последние 10-20% пространства диска, так как они нужны для корректной работы команды «TRIM». Для работы ей нужно свободное пространство для перегруппировки данных (так же, как для функции дефрагментации). Общее правило — чем больше свободного места — тем быстрее работает устройство.

На данный момент SSD диск идеально подходит в роли системного раздела, на котором установлена операционная система и программы. Данные и вся работа над ними должна (по возможности) проходить на HDD диске. Также твердотельные диски могут эффективно использоваться на серверах для кеширования СТАТИЧНЫХ данных.

**При достижении критической массы сбойных ячеек накопителя**, качественно выполненная прошивка чипа контроллера может полностью заблокировать SSD диск для функций записи и **перевести его в режим «только чтение»**, что гарантирует сохранность данных пользователя (возможность сделать резервную копию данных) до полного выхода устройства из строя.

Самым главным отличием различных типов 3D NAND памяти является их долговечность!

# Принцип работы и износа 3D NAND памяти

Память NAND состоит из ячеек транзистора с плавающим затвором, которые сохраняют заряженное состояние при отсутствии источника питания. Плавающие затворы содержат электроны, а заряженное состояние представлено двоичным разрядом 0 и разряженным состоянием 1. Двоичный бит 0 представляет данные, хранящиеся в памяти NAND.

# Принцип работы NAND-памяти

Ячейки присутствуют в сетке, известной как блок. Отдельная строка в блоке называется страницей и поддерживает размеры 2К, 4К, 8К и 16К. Каждый блок содержит 128-256 страниц, поэтому приблизительный его размер варьируется от 256Кб до 4Мб.

Флэш-память сохраняет информацию путем захвата электронов в ячейках. Присвоенный ячейке заряд и определяет наличие данных. Процесс ввода-вывода электронов оказывает негативное воздействие на структуру ячейки, а часть электронов «застревает». Эти электроны создают отрицательный заряд, уменьшая диапазон напряжений, доступных для представления данных. Чем меньше становится этот диапазон, тем труднее твердотельным накопителям выполнять запись и проверять её достоверность.

**Накопление электронов особенно разрушительно при более высоких битовых плотностях**. MLC-память должна различать четыре возможных значения в пределах уменьшающегося окна напряжений, но TLСпамять должна отслеживать в два раза больше значений, а QCL — в четыре раза больше. В результате память с более плотной компоновкой «прожигается» быстрее, тем самым имея меньший ресурс.

Продолжение цикла записи в конечном итоге приводит к тому, что ячейки становятся ненадёжными. Эти ячейки в дальнейшем удаляются и заменяются флэш-памятью, извлеченной из «резервной области» накопителя. Этот резерв новой области флэш-памяти гарантирует, что твердотельный накопитель сохраняет доступную для пользователя ёмкость хранилища, даже если отказ отдельных ячеек выводит из строя часть памяти. В конечном итоге и этот резерв истощается, и накопитель начинает выходить из строя.

Рассматривая типы памяти подробнее, нельзя не упомянуть относительно свежую разработку корпорации Intel — память 3D XPoint и накопители Intel Optane на её основе.

## Надёжность SSD

Казалось бы, нет движущихся частей – все должно быть очень надежно. Это не совсем так. Любая электроника может сломаться, не исключение и SSD. С низким ресурсом MLC-чипов ещё можно как-то бороться коррекцией ошибок ECC, резервированием, контролем за износом и перемешиванием блоков данных. Но **самый большой источник проблем – контроллер и его прошивка**. По причине того, что контроллер физически расположен между интерфейсом и микросхемами памяти, вероятность его повреждения в результате сбоя или проблем с питанием очень велика. При этом сами данные, в большинстве случаев сохраняются.

Помимо физических повреждений, при которых доступ к данным пользователя невозможен, существуют логические повреждения, при которых нарушается доступ к содержимому микросхем памяти. Любая, даже незначительная ошибка, баги в прошивке, может привести к полной потере данных. Структуры данных очень сложные. Информация «размазывается» по нескольким чипам, плюс чередование, делают восстановление данных сложной задачей.

В таких случаяхвосстановить накопител[ь](https://storelab-rc.ru/) помогает прошивка контроллера с низкоуровневым форматированием, когда заново создаются служебные структуры данных. Производители стараются постоянно дорабатывать микропрограмму, исправлять ошибки, оптимизировать работу контроллера. По этому, рекомендуется периодически обновлять прошивку накопителя для исключения возможных сбоев.

## Безопасность SSD

В SSD накопителе, как и в HDD, данные не удаляются сразу после того, как файл был стёрт из ОС. Даже если переписать файл по верху нулями – физически данные еще остаются, и если чипы флеш-памяти достать, и считать на программаторе – можно найти 4кб фрагменты файлов. Полное стирание данных стоит ждать тогда, когда на диск будет записано данных равное количеству свободного места + объем резерва (примерно 4 Гб для 60Гб SSD). Если файл попадёт на «изношенную» ячейку, контроллер ещё не скоро перезапишет её новыми данными.

Основные принципы, особенности, отличия в восстановлении данных с SSD и USB Flash накопителей. [Восстановление данных с SSD](https://storelab-rc.ru/ssd.htm) накопителей достаточно трудоёмкий и долгий процесс по сравнению с портативными flash накопителями. Процесс поиска правильного порядка, объединения результатов и выбора необходимого сборщика (алгоритм/программа полностью эмулирующая работу контроллера SSD накопителя) для создания образа диска не лёгкая задача.

Связанно это в первую очередь с увеличением числа микросхем в составе SSD накопителя, что во много раз увеличивает число возможных вариантов действий на каждом этапе восстановления данных, каждое из которых требует проверки и специализированных знаний. Так же, в силу того, что к SSD предъявляются значительно более жесткие требования по всем характеристикам (надёжность, быстродействие и т.д.), чем к мобильным флеш накопителям, технологии и методики работы с данными, применяемые в них, достаточно сложны, что требует индивидуального подхода к каждому решению и наличию специализированных инструментов и знаний.

## Оптимизация SSD

1. Для того, чтобы диск прослужил вам долго, нужно всё, что часто меняется (временные файлы, кеш браузера, индексирование) необходимо перенести на HDD, отключить обновление времени последнего доступа к папкам и каталогам (fsutil behavior set disablelastaccess 1). Отключить в ОС дефрагментацию файлов.
2. Перед установкой на SSD Windows XP, при форматировании диска рекомендуется выполнить «выравнивание» разделов кратным степени двойки (например, утилитой **diskpart**), иначе SSD придется делать 2 чтения вместо одного. Кроме этого у Windows XP есть некоторые проблемы с поддержкой секторов более 512кб (в SSD по умолчанию используется 4кб) и вытекающие отсюда проблемы с производительностью. Windows Vista,

Windows 7, последние версии Mac OS и Linux выравнивают диски уже правильно.

1. Обновить прошивку контроллера, если старая версия не знает команду TRIM. Установить последние драйвера на SATA контроллеры. Например, если у вас контроллер от Intel, вы можете на **10-20%** увеличить производительность, включив режим **ACHI** и установив Intel Matrix Storage Driver в операционной системе.
2. Не следует использовать последние 10-20% свободного пространства от раздела, потому что, это может отрицательно сказаться на производительности. Это особенно важно, когда работает **TRIM**, поскольку ему необходимо пространство для перегруппировки данных: для примера, похоже, работают утилиты дефрагментации, ведь им тоже нужно не менее **10**% процентов от объема диска. Поэтому очень важно следить за данным фактором, ведь из-за небольшого объема SSD они очень быстро заполняются.

## Преимущества SSD

* высокая скорость чтения любого блока данных не зависимо физического от расположения (более 200 Мб/с);
* низкое энергопотребление **при чтении** данных с накопителя

(приблизительно на 1 Ват ниже, чем у HDD);

* пониженное тепловыделение (внутреннее тестирование в компании Intel показало, что ноутбуки с SSD нагреваются на 12.2° меньше чем аналогичные с HDD, также тестированием установлено, что ноутбуки с SSD и 1 GB памяти в распространенных бенчмарках не уступают моделям с HDD и 4 GB памяти);
* бесшумность и высокая механическая надёжность.

## Недостатки SSD

- высокое энергопотребление **при записи** блоков данных, энергопотребление растёт с ростом объёма накопителя и интенсивностью изменения данных; - низкая ёмкость и высокая стоимость за гигабайт по сравнению с HDD; - ограниченное число циклов записи.

**Еще о некоторых минусах:**

* Все SSD имеют ограниченный ресурс по количеству записываемой информации. Из-за особенностей архитектуры флэш-памяти и методов записи происходит деградация ячеек памяти. Со временем это приводит к уменьшению доступного объёма и отказу накопителя. Но не всё так плохо, как кажется на первый взгляд. Даже относительно бюджетные модели могут обладать ресурсом перезаписи в районе 200 циклов, не говоря уже о более дорогих моделях. Несложно подсчитать, что твердотельный накопитель ёмкостью в 500 ГБ и ресурсом перезаписи в 200 циклов позволит вам на протяжении 5 лет ежедневно записывать и перезаписывать 55 гигабайт информации, что немало.
* Несмотря на стремительное развитие на протяжении уже 10 лет, цена 1 гигабайта памяти SSD всё ещё значительно превышает цену 1 гигабайта памяти классического жёсткого диска. Разница составляет от двух раз у бюджетных моделей до десяти и более раз по сравнению с моделями верхнего ценового сегмента.

**Заключение**

В связи с высокой стоимостью SSD дисков и небольшим объёмом памяти использовать их для хранения динамических данных нецелесообразно. Зато они отлично подойдут в качестве системного раздела, на который инсталлируется ОС и на серверах для кэширования статичных данных.

При выборе SSD для своей системы вы можете столкнуться со следующими форматами:

* **mSata** — несколько устаревший формат низкопрофильной платы, предназначенный для ноутбуков, планшетов и портативной техники; использует подключение через интерфейс SATA.
* **2.5 дюйма** — всем знакомый по портативным жёстким дискам формат, который используется как в настольных системах, так и в ноутбуках; использует подключение через интерфейс SATA.
* **M.2** — современный формат низкопрофильной платы, позволяющий осуществлять подключение через специальный слот M.2; использует подключение как через интерфейс SATA, так и через PCI-Express.
* **PCI-Express AIC** — карты расширения для слотов PCI-Express, предназначенные для настольных компьютеров, рабочих станций и серверов.
* **U.2** — формат 2.5-дюймовых накопителей, разработанный для использования через интерфейс PCI-Express при помощи специального коннектора малого формата.

**Интерфейс подключения SSD напрямую влияет на его скорость**. Максимальная скорость передачи данных твердотельного накопителя ограничена типом подключения.

SSD потребительского сегмента используют подключение через интерфейсы SATA и PCI-Express:

* **SATA/mSATA** — обеспечивает передачу данных на скоростях до 6

Гбит/сек (SATA III).

* **M.2 SATA** — обеспечивает передачу данных на скоростях до 6 Гбит/сек (SATA III).
* **M.2 NVMe** — обеспечивает передачу данных по линиям PCI-Exprees на скоростях до 31.5 Гбит/сек (PCI-Express 3.0 x4).
* **AIC NVMe** — обеспечивает передачу данных на скоростях до 31.5

Гбит/сек (PCI-Express 3.0 x4).

Наиболее распространёнными форм-факторами современных потребительских твердотельных накопителей являются 2.5-дюймовый и М.2 форматы.

2.5-дюймовые SATA SSD можно охарактеризовать как некий переходный вариант в цепи развития твердотельных накопителей. Они позволяют установить новый тип накопителя в старую систему, тем самым получив все плюсы использования SSD без замены ключевых компонентов ПК.

**М.2** — это современный стандарт, доля которого на рынке стремительно растет. Твердотельные накопители формата М.2 могут подключаться как по линиям SATA, так и по линиям PCIe, используя протокол NVMe. М.2 отличается от прочих форм-факторов комплексностью. Давайте рассмотрим его подробнее.

Стандарт M.2 был разработан консорциумами PCI-SIG и SATA-IO. Изначально он назывался NGFF — Next Generation Form Factor (ФормФактор Следующего Поколения) и был переименован в М.2 в 2013 году. В форм-факторе М.2 выпускается большое количество карт расширения, такие как:

* модули Wi-Fi, Bluetooth и NFC.
* модули WiGig и Wireless WAN (WWAN).
* модули спутниковый навигации и цифрового радио.
* модули SSD.

Стандарт М.2 включает в себя:

* **12 вариантов подключений** (keying options), обозначающихся литерами от A до M.  **4 варианта ширины устройств** (12, 16, 22 и 30 мм).
* **8 вариантов длины** (16, 26, 30, 38, 42, 60, 80 и 110 мм).  **7 вариантов толщины** (высоты) устройств.

Твердотельные накопители М.2 за счёт своей компактности отлично подходят для установки как в обычные модели ноутбуков, так и в ультрабуки. **Самыми распространенными вариантами M.2 SSD являются**:

* **2280** — имеют длину 80 мм и ширину 22 мм
* **2260** — имеют длину 60 мм и ширину 22 мм
* **2242** — имеют длину 42 мм и ширину 22 мм
* **2230** — имеют длину 30 мм и ширину 22 мм
* **Твердотельные накопители M.2 NVMe** отличаются компактными размерами и имеют все преимущества подключения через интерфейс PCI-Express — они обладают более высокими скоростями и более низкими задержками, чем SATA SSD. Их главным минусом является снижение производительности при перегреве (троттлинг), которое может возникать во время длительной высокой нагрузки.
* Однако при сравнении производительности NVMe SSD и традиционных SATA SSD в повседневной работе и реальных сценариях использования наблюдается лишь небольшой разрыв. Всё дело в том, что в какой-то момент скорость считывания данных сравнивается со скоростью их обработки, и дальнейший рост характеристик не приводит к улучшению производительности. В таких

случаях ситуацию может улучшить дальнейшая оптимизация приложений и увеличение производительности остальных компонентов системы.

Разновидность твердотельных дисков — «RAM SSD» накопители. Это гибридные устройства, использующие для хранения информации энергозависимые чипы, полностью идентичные тем, что используются в модулях оперативной памяти компьютера. Они обладают сверхбыстрой скоростью доступа к данным, скоростью чтения и записи и могут с успехом применяться для ускорения работы больших баз данных и там, где нужно пиковое быстродействие. Подобные системы оснащаются аккумуляторами для поддержания функционирования при отсутствии электроэнергии, а более дорогие модели — системами резервного копирования, когда данные копируются на HDD носитель.

