МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет информационных технологий

А. Р. Гафиятуллин, Н. А. Радеев

НОРМАЛЬНЫЕ ОТВЕТЫ НА НЕНОРМАЛЬНЫЕ ВОПРОСЫ ЗАЧЕТА ПО ЭВМ И ПУ

ЧАСТЬ ПЕРВАЯ И ПОСЛЕДНЯЯ

Учебное пособие

Новосибирск
2018

УДК ∞
ББК ∞
П ∞

Рецензент:
канд. инф.-мемологических наук Н. С. Каминский

Гафиятуллин А. Р., Радеев Н. А.

П ∞ Нормальные ответы на ненормальные вопросы зачета по ЭВМ и ПУ : учеб. пособие / А. Р. Гафиятуллин, Н. А. Радеев ; Новосиб. гос. ун-т. – Новосибирск, 2018. – Ч. 1. и последняя.

ISBN 8-800-555-35-35

В пособии излагаются основы организации ЭВМ, периферийных устройств и способов сдачи зачета. Оно не написано на основе лекционной и практической работы авторов с преподавателями ФИТ НГУ. Цель пособия – ознакомить читателя с основными понятиями и начальными результатами организации ЭВМ, периферийных устройств и способов сдачи зачета.

Предназначено для студентов факультетов информационных технологий, математических, естественнонаучных и философских факультетов университетов, а также для всех самостоятельно изучающих основы организации ЭВМ, периферийных устройств и способов сдачи зачета.

УДК ∞
ББК ∞

ISBN 8-800-555-35-35

Посвящается лекциям
по ЭВМ и ПУ

Благодарности

Интернету за помощь в подготовке рукописи и моральную поддержку.

Гафиятуллину А. Р., Радееву Н. А., Каминскому Н. С. и многим другим за обсуждение и ценные советы.

Студентам факультета информационных технологий НГУ, на которых в течении 5 лет “обкатывались” содержание, стиль изложения, очепятки в методичке и список вопросов к зачету 2016 года.

1. УЗКИЕ МЕСТА АРХИТЕКТУРЫ ФОН НЕЙМАНА И ПУТИ ЕЕ УСОВЕРШЕНСТВОВАНИЯ.
    1. **ПРОБЛЕМА:** Хранение программы и данных в одной памяти и последовательное выполнение команд ограничивает скорость работы компьютера пропускной способностью канала передачи этих команд и данных.

       РЕШЕНИЕ:

       Ускорение доступа к памяти:

        1. Аппаратная и программная предвыборка команд и данных (Ссылочка)
        2. Дополнительные уровни иерархии памяти (кэш, регистр);
        3. Виртуальная память (Тыц);
        4. Большой регистровый файл (Определение);
        5. Высокоскоростные шины;

       Ускорение выполнения команд:

        6. Упрощение набора команд;
        7. Конвейеризация;
        8. Параллелизм выполнения программ.

       Источник: методичка, стр. 11 - 12.

2. ИЗВЕСТНЫЕ КЛАССЫ АРХИТЕКТУР КОМПЬЮТЕРОВ.

   ОТВЕТ: CISC, RISC, VLIW (наследник RISC. Itanium и Эльбрус это VLIW) CISC, RISC, VLIW

   CISC:

   - нефиксированное значение длины команды
   - арифметические действия кодируются в одной команде;

- небольшое число регистров, каждый из которых выполняет строго определённую функцию.

RISC:

- Фиксированная длина машинных инструкций (например, 32 бита) и простой формат команды. *В киске команды разной длины и это хуже для конвейера, так как команды выполняются за различное число тактов и конвейер будет стоять и ждать, пока выполнится какая-нибудь сложная команда, в риске такой лажи нет.*
- Специализированные команды для операций с памятью — чтения или записи. Операции вида Read-Modify-Write («прочитать-изменить-записать») отсутствуют. Любые операции «изменить» выполняются только над содержимым регистров (т. н. архитектура load-and-store). *Простыми словами, тут сначала загружаешь в регистр, преобразовываешь и затем выгружаешь в память, в киске всё это можно сделать одной командой например.*
- Большое количество регистров общего назначения (32 и более).
- Отсутствие поддержки операций вида «изменить» над укороченными типами данных — байт, 16-разрядное слово. *Например, если регистр 64 бита, то в риске нельзя изменить первые 16 бит, или первые 32 бита, только через танцы с бубном. В киске это можно, там в зависимости от первой буквы в имени регистра (типа RAX, EAX) мы меняем те или иные биты.*
- Отсутствие микропрограмм внутри самого процессора. То, что в CISC-процессоре исполняется микропрограммами, в RISC-процессоре исполняется как обыкновенный (хотя и помещенный в специальное хранилище) машинный код, не отличающийся принципиально от кода ядра ОС и приложений.

VLIW:

Характеризуется тем, что одна инструкция процессора содержит несколько операций, которые должны выполняться параллельно.

В суперскалярных процессорах также есть несколько вычислительных модулей, но задача распределения работы между ними решается аппаратно. Это сильно усложняет устройство процессора, и может быть чревато ошибками. В процессорах VLIW задача распределения решается во время компиляции и в инструкциях явно указано, какое вычислительное устройство какую команду должно выполнять. Т.е. от аппаратного усложнения ушли к программному.

VLIW можно считать логическим продолжением идеологии RISC, расширяющей её на архитектуры с несколькими вычислительными модулями. Так же, как в RISC, в инструкции явно указывается, что именно должен делать каждый модуль процессора. Из-за этого длина инструкции может достигать 128 или даже 256 бит.

Подход VLIW сильно упрощает архитектуру процессора, перекладывая задачу распределения вычислительных устройств на компилятор.

Из-за большого количества пустых инструкций для простаивающих устройств программы для VLIW-процессоров могут быть гораздо длиннее, чем аналогичные программы для традиционных архитектур.

Программирование вручную, на уровне машинных кодов для VLIW-архитектур, является достаточно сложным. Приходится полагаться на оптимизацию компилятора.

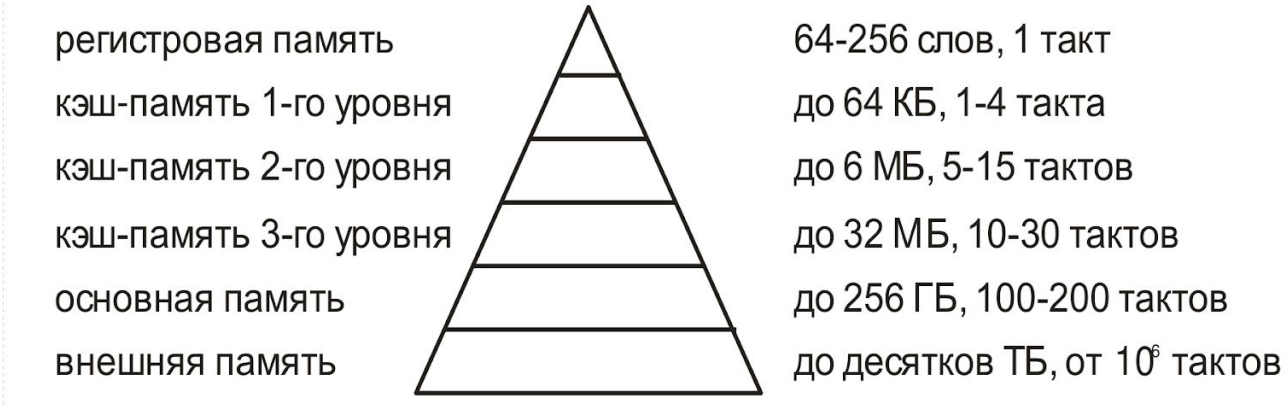
3. СРАВНИТЕ СПОСОБЫ УПРАВЛЕНИЯ ИЕРАРХИЧЕСКОЙ ПАМЯТЬЮ.

ОТВЕТ:

Идея **иерархической (многоуровневой)** организации памяти заключается в использовании на одном компьютере нескольких уровней памяти, которые характеризуются разным временем доступа к памяти и объемом памяти. (Время доступа к памяти это время между операциями чтения/записи, которые выполняются по случайным адресам.) Основой для иерархической организации памяти служит принцип **локальности ссылок во времени и в пространстве**.

1. **Локальность во времени** состоит в том, что процессор многократно использует одни и те же команды и данные.
2. **Локальность в пространстве** состоит в том, что если программе нужен доступ к слову с адресом A, то скорее всего, следующие ссылки будут к адресам, расположенным поблизости с адресом A.

Типичная организация иерархической памяти в современных компьютерах:



Чем выше уровень - тем меньше памяти, тем она быстрее и дороже за единицу.

В современном компьютере существует **два механизма** управления иерархической памятью, которые задают правила пересылок фрагментов кода программы и данных между уровнями памяти:

1. Механизм **кэширования**;
2. Механизм **виртуальной памяти**.

Собственно, и **сравним** их:

	Кэширование	**Виртуальная память**
Для чего нужна?	Иллюзия большой, быстрой и одноуровневой памяти	Иллюзия большой, быстрой и одноуровневой памяти(по сути все адресное пространство, хотя физической, очевидно, намного меньше). Помогает реализовывать многозадачность ОС.
Что связывает?	Кэш-память и оперативку	Оперативку и внешнюю память
Как реализовано в компах? Кто управляет?	Целиком аппаратно кэш-контроллером	Совместно процессором и операционной системой
Из чего обычно состоит?	Кэш-строки(на них делится как кэш-память, так и оперативная память)	Страницы(на них делится как виртуальная память,так и физическая память, потому что виртуальные страницы хранятся в физических)
Способ адресации?	кэш-память сама по себе не попадает в адресное пространство, а адрес байта в оперативке особо	Адрес виртуальной (физической) страницы состоит из номера страницы и смещения (адреса

	интерпретируется при отображении в кэш-память. Как это происходит, вообще, зависит от алгоритма отображения(это отдельный 4-ый вопрос), но обычно всегда: ведущие биты адреса байта - тег строки, а младшие - смещение байта в строке относительно ее начала. Может быть еще номер множества(средние биты после смещения). Это однозначно(с точностью до банка) задает место, в которое будет отображена строка в кэш.	относительно начала страницы). Виртуальное адресное пространство описывается двумя таблицами: таблицей страниц и картой диска. Таблица страниц устанавливает соответствие виртуальных и физических адресов страниц. Карта диска содержит информацию о расположении страниц во внешней памяти.
Способы улучшения работы механизма?	Аппаратная и программная предвыборка данных.	Многоуровневые таблицы страниц; Буферы быстрого преобразования адресов (TLB).

Источники: методичка, стр. 14 - 42 (стр. 39 - 41 - алгоритм преобразования виртуальных адресов в физические)

4. СРАВНИТЕ АЛГОРИТМЫ ОТОБРАЖЕНИЯ ДАННЫХ В КЭШ-ПАМЯТЬ.

   Методичка стр. 27.

   Существуют три схемы отображения адресов из оперативки в кэш: ***прямая***, ***ассоциативная*** и ***множественно-ассоциативная.***

   - **Прямое отображение**: адрес ячейки в оперативке состоит из тэга, индекса и смещения внутри блока. Тэг - старшая часть адреса, индекс - средняя, смещение - младшая. Оперативка отображается в кэш по правилу S = A mod C, где S -

адрес строки кэша, A - адрес в оперативке, C - число строк в кэше. То есть каждая C-ая ячейка оперативки претендует на одну и ту же ячейку в одной и той же строке кэша. Таким образом, в строке кэша хранится тег той ячейки, которая в данный момент решила заехать к кэш и собственно данные этой ячейки. Младшая часть адреса ячейки оперативной памяти это смещение, которое показывает положение байта в строке кэша, так как в строке может помещаться больше одного байта (а может и нет, но это очень убогий кэш какой-то) (см. Картинку в методичке на стр. 28) Если на занятую ячейку претендует другая память, то возникнет буксование, потому что нужно вытеснить предыдущий элемент из кэша и обратиться в оперативку за новым.

- **Ассоциативное отображение**: любой блок памяти может быть записан в любую строку кэша, то есть, тут буксования нет, разве что при полном заполнении кэша. При ассоциативном отображении адрес байта в оперативке представляется как тег и смещение внутри кэш-строки, то есть нет индекса, который определял в прямом отображении номер строки. Так как теперь строки кэша не пронумерованы, то для того, чтобы понять, лежат ли данные в кэше, нужно сравнить тег ячейки из оперативки со всеми тегами строк кэша. Это занимает больше времени, чем при прямом отображении, где нам сразу известно, в какой строке наши данные нужно искать. Таким образом, избавились от буксования, но замедлили работу кэша вцелом. (методичка стр.29)
- **Множественно-ассоциативное отображение**: суть в том, что теперь по одному индексу существуют несколько строк кэша. Обычно их количество это степени двойки 2, 4, 8, 16, но легко встретить и 12 например. Грубо говоря, кэш как бы разбивается на несколько кэшей прямого отображения. Один слой

такого кэша это *банк*, количество банков - это *степень ассоциативности* кэша. Адрес состоит так же, как в прямом отображении, из тега, номера строки и смещения. Решение, в какой банк попадут данные, принимает кэш-контроллер, а затем уже в пределах одного банка всё работает как при прямом отображении. Буксование возникнуть может, но для этого нужно заполнить ячейку во всех банках, что случается, в общем случае, случается реже, чем заполнение одной ячейки в прямом отображении.

5. СПОСОБЫ ПОДДЕРЖАНИЯ КОГЕРЕНТНОСТИ ДАННЫХ В КЭШ-ПАМЯТИ.

ПРОБЛЕМА:

Когерентность кэша - свойство кэшей, означающее целостность данных, хранящихся в локальных кэшах для разделяемого ресурса. Проблема когерентности кэш-памяти характерна для многоядерных(многопроцессорных) систем. Например, если одно ядро(процессор) изменило данные и записало их обратно в кэш, но не в оперативку, а эти же данные потребовались второму ядру(процессору), то оно, запросив их у оперативки, вообще говоря рискует получить неактуальные данные. Проблема, о которой идет речь, возникает из-за того, что значение элемента данных в памяти, хранящееся в двух разных ядрах(процессорах), доступно этим ядрам(процессорам) только через их индивидуальные кэши.

Когерентность определяет поведение чтений и записей в одно и то же место памяти. Кэш называется когерентным, если выполняются следующее условие:

Информирование о записи. Изменение данных в любом из кэшей должно быть распространено на другие копии этих данных (этой линии кэша) в соседних кэшах.

РЕШЕНИЕ:

1. **Когерентность с использованием справочника.** Информация о состоянии блока физической памяти содержится только в одном месте, называемом справочником (физически справочник может быть распределен по узлам системы).

   Протокол на основе “полного справочника” является наиболее простым из протоколов на основе директорий. Здесь присутствует единый централизованный справочник, хранящийся в основной памяти, который поддерживает информацию обо всех кэшах. Справочник содержит множество записей, описывающих каждую кэшируемую ячейку ОП. Обращение к справочнику производится каждый раз, когда один из процессоров изменяет копию такой ячейки в своей локальной памяти. В этом случае информация из справочника нужна для того, чтобы аннулировать или обновить копии измененной ячейки (или всей строки, содержащей эту ячейку) в других локальных кэшах, где присутствует эта же строка. Для каждого блока ОП в справочнике выделяется одна запись, хранящая указатели на копии данной строки. Кроме того, в каждой записи выделен один бит модификации (D), показывающий, является ли копия “грязной”- (D = 1 - dirty) или “чистой”-(D = 0 - clean), то есть изменялось ли содержимое строки в кэш-памяти после того, как она была туда загружена. Этот бит указывает, имеет ли право процессор производить запись в данную строку.

2. **Когерентность с использованием отслеживания.** Каждый кэш, который содержит копию данных некоторого блока физической памяти, имеет также

соответствующую копию служебной информации о его состоянии. Централизованная система записей отсутствует. Обычно кэши расположены на общей (разделяемой) шине и контроллеры всех кэшей наблюдают за шиной (просматривают ее) для определения того, не содержат ли они копию соответствующего блока.

Протокол **MESI:**

Протокол MESI реализован в современных многоядерных процессорах Intel и архитектуре PowerPC. Каждая строка кэша, согласно протоколу MESI, может находиться в одном из **четырех состояний**:

Modified - строка в кэше была изменена (отлична от основной памяти) и новое значение доступно только в данном кэше;

Exclusive - значение строки совпадает со значением в основной памяти, но его нет ни в одном другом кэше;

Shared - значение строки совпадает со значением в основной памяти и может присутствовать в других кэшах;

Invalid - строка кэша содержит недостоверную информацию.

3. **Перехват.** Когда из какого-либо одного кэша данные переписываются в оперативную память, контроллеры остальных получают сигнал об этом изменении ("перехватывают" информацию об изменении данных) и, если необходимо, изменяют соответствующие данные в своих кэшах.

Источники: Википедия, методичка Донецкого универа.

6. ПРИВЕСТИ ПРИМЕРЫ ЗАДАЧ, НА КОТОРЫХ ЭФФЕКТИВНО И НЕЭФФЕКТИВНО РАБОТАЮТ ВСЕ ВИДЫ АЛГОРИТМОВ ОТОБРАЖЕНИЯ ДАННЫХ В КЭШ-ПАМЯТЬ.

   Неэффективно: при случайном обходе массива кэш-контроллер не может предугадать порядок обращения к элементам. И это не зависит от алгоритма отображения данных в кэш, так как кэш-контроллер попросту не знает, какие данные понадобятся в будущем. Из-за этого на каждом шаге обхода с огромной вероятностью будет происходить кэш-промах, то есть данные не будут обнаружены в кэше.

   Эффективно: при прямом обходе массива кэш-контроллер скорее всего поймёт, какие данные понадобятся в будущем и произойдёт предвыборка данных в кэш. Более того сработает блочная загрузка и загрузится сразу целый блок данных. Независимо от алгоритма отображения данных в кэш, кэш-промахи будут случаться редко.

   Источник: методичка и лабы 8 и 9.

7. СРАВНИТЬ ЦЕНУ ПРОМАХА В КЭШ-ПАМЯТИ И В ВИРТУАЛЬНОЙ ПАМЯТИ.

	Кэш-память	**Виртуальная память**
Скорость доступа при попадании H:	Если брать среднее количество тактов по всем уровням кэша, то порядка **15** тактов.	Примерно **200** тактов.
Обращение за недостающими данными в более низкие уровни памяти L:	Обращается в оперативную память. Если допустить, что необходимые данные уже в оперативной памяти, то	Обращается в жесткий диск. Порядка 10^6 тактов.

	доступ к ней займет порядка **200** тактов.	
Шанс промаха R:	Около 10%	0.001%

Введем величину V, отражающую цену промаха. Понятно, что она прямо-пропорциональна шансу промаха R, потому что чем чаще происходит промах, тем медленнее работает память. V прямо-пропорциональна $\frac{L}{H}$, потому что чем медленнее более низкий уровень памяти по сравнению с более высоким, тем больший вклад вносится в цену промаха. Почему не используются такие показатели памяти, как частота обращения и объем? Эти показатели “съедаются” шансом промаха R. Действительно, чем выше шанс промаха, тем выше частота обращений(просто проводится больше “экспериментов” и у быстрой памяти нет большого запаса времени на оптимизацию хит рейта) и тем меньше объем(данных отобразить в быструю память надо много, а ее мало, вот и происходят частые промахи).

Таким образом, $V = R * \frac{L}{H}$

V кэш-памяти: $V = 10 * \frac{200}{15} \approx 133$

V виртуальной памяти: $V = 0.001 * \frac{1000000}{200} = 5$

Таким образом, **цена промаха кэш-памяти выше, чем цена промаха виртуальной памяти**, во многом благодаря низкому мисс рейту виртуальной памяти.

Источники: брошюры университета Айовы по кэшу и по виртуальной памяти, пирамида иерархии памяти сверху, а так же здравый смысл.

8. ПУТИ УМЕНЬШЕНИЯ ВРЕМЕНИ ДОСТУПА К ПАМЯТИ.
    - Предвыборка данных - механизм загрузки данных из оперативки в кэш еще до того как они реально понадобятся процессору, уменьшает время простоя

процессора. То есть в каком-то смысле, процессор быстрее обращается к памяти. Бывает аппаратной и программной.

- Следует писать программы так, чтобы доступ к используемой памяти по возможности был последовательным, тогда кэш-контроллер сможет совершить предвыборку данных. Нужно программировать с учётом иерархичности памяти, использовать оптимизации компилятора там, где это возможно. Избегать перевыделений памяти, так как эта операция занимает много времени. Часто используемые переменные, например итерационные переменные циклов, следует класть в регистры.

  Источники: методичка стр.23, отсебятина.

9. КОМАНДНЫЙ КОНВЕЙЕР. ПРИМЕР КОМАНДНОГО КОНВЕЙЕРА. СПОСОБЫ УВЕЛИЧЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ КОНВЕЙЕРА. ПРИЧИНЫ ПРИОСТАНОВКИ КОНВЕЙЕРА И ТЕХНИКА ИХ ПРЕОДОЛЕНИЯ.

   ОТВЕТ:

   Конвейеризация – это техника, в результате которой задача или команда разбивается на некоторое число подзадач, которые выполняются последовательно. Каждая подкоманда выполняется на своем логическом устройстве. Все логические устройства (ступени) соединяются последовательно таким образом, что выход i-ой ступени связан с входом (i+1)-ой ступени, все ступени работают одновременно. Множество ступеней называется конвейером. Выигрыш во времени достигается при выполнении нескольких задач за счет параллельной работы всех ступеней, вовлекая на каждом такте новую задачу или команду.

   Таким образом, выполнение команды разделяется на несколько стадий, каждая из которых выполняется на соответствующей ступени конвейера. **Типичный набор стадий выполнения команды:** 1) Выборка команды

2) Декодирование команды

3) Выборка и загрузка операндов

4) Выполнение операции

5) Сохранение результатов в память

Причины низкой производительности или приостановки конвейера и техника их преодоления:

1. **Зависимости по данным**(когда команда уже может получить операнды, но над этими операндами работает выполняющаяся команда):
    1. **Data forwarding**- это техника ускоренной передачи результата одной операции на другую операцию через процессорные регистры без записи результата в ячейку памяти;
    2. **Разнесение зависимых по данным команд**, заполняя пространство между ними другими независимыми от них командами.
2. **Конфликты по ресурсам**(на некоторой стадии команда должна обратиться к некоторому ресурсу процессора, который занят другой командой):
    1. Для устранения зависимостей по ресурсам на уровне микроархитектуры обычно используется **дублирование ресурсов.** Например, в процессоре может быть несколько функциональных устройств, которые работают одновременно. Один регистровый файл может быть разбит на два (или даже продублирован), доступ к которым осуществляется независимо. На уровне кода конфликтующие по ресурсам команды обычно разносятся компилятором на некоторое расстояние, и между ними вставляются другие, независящие от них команды.
3. **Зависимости по управлению**, вызванные командами перехода(на конвейер попала команда, которая не должна быть выполнена из-за обработанного

только что ветвления в программе - придется делать сброс конвейера, начиная работу с команды, которая была выбрана ветвлением):

1. **Механизм раннего обнаружения и предсказания переходов:**
    1. **Статические методы** используют информацию из кода программы, специально выработанную компилятором. При использовании этого способа процессор делает однозначный вывод о срабатывании команды перехода по ее виду. Статическое предсказание срабатывает **на стадии декодирования команды**;
    2. **Динамический** (учитывает предысторию выполнения программы, она записывается в таблицу истории переходов (Branch History Table (BHT)). В строке таблицы для команды условного перехода содержится целевой адрес и бит, который указывает, был ли сделан переход, когда команда встретилась последний раз. Если прогноз не верен, то бит в таблице меняется. Динамическое предсказание срабатывает уже **на стадии выборки команды**, т.к. команды перехода в таблицах истории идентифицируются по своему адресу.

Источник: методичка, стр. 44 - 50.

10. УСЛОВНЫЕ КОМАНДЫ: ЧТО ЭТО ТАКОЕ И ЦЕЛИ ИХ ВВЕДЕНИЯ В СИСТЕМУ КОМАНД.

Кома́нда перехо́да — команда процессора, которая нарушает непрерывную последовательность исполнения команд, вынуждая выбирать и исполнять последующие команды с произвольно заданного адреса. Используется для организации условных операторов, циклов, для связи с подпрограммами. Исполнение команды перехода в современных микропроцессорах чревато потерями производительности из-за простоев конвейера. (ист. википедия)

В зависимости от своего операнда, команда условного перехода может выбрать инструкцию в программе, которую требуется выполнять далее. Таким образом можно делать циклы, итерационные процессы и всякое такое. (ист. Методичка стр. 6)

11. СРАВНИТЬ РЕАЛИЗАЦИЮ КОНВЕЙЕРА В CISC И RISC-АРХИТЕКТУРАХ.

ОТВЕТ:

RISC:

Простота структуры и небольшой набор команд позволяет реализовать полностью их аппаратное выполнение и эффективный конвейер при небольшом объеме оборудования. Арифметику RISC - процессоров отличает высокая степень дробления конвейера. Этот прием позволяет увеличить тактовую частоту (значит, и производительность) компьютера; чем более элементарные действия выполняются в каждой фазе работы конвейера, тем выше частота его работы. RISC - процессоры с самого начала ориентированны на реализацию всех возможностей ускорения арифметических операций, поэтому их конвейеры обладают значительно более высоким быстродействием, чем в CISC - процессорах. Поэтому RISC - процессоры в 2 - 4 раза быстрее имеющих ту же тактовую частоту CISC - процессоров с обычной системой команд и высокопроизводительней, несмотря на больший объем программ, на (30 %).

1. Каждая операция разбивается на однотипные простые этапы, которые выполняются параллельно;
2. Каждый этап занимает 1 такт, в т.ч. декодирование.

CISC:

Конвейеризация потенциально применима к любой процессорной архитектуре, независимо от набора команд и положенных в ее основу принципов. Даже самый первый x86-процессор, Intel 8086, уже содержал своеобразный примитивный

"двухстадийный конвейер" - выборка новых инструкций (FETCH) и их исполнение осуществлялись в нем независимо друг от друга. Однако реализовать что-то более сложное для CISC-процессоров оказалось трудно: декодирование неоднородных CISC-инструкций и их очень сильно различающаяся сложность привели к тому, что конвейер получается чересчур замысловатым, катастрофически усложняя процессор. Поэтому, для реализации эффективного конвейера, CISC-команды динамически декодируются в RISC-команды, что добавляет еще одну стадию конвейера.

Источники: IXBT, Компьютерра.

12. СРАВНИТЬ СИСТЕМУ КОМАНД CISC И RISC-АРХИТЕКТУРЫ.

CISC:

· большое число различных по формату и длине команд;

· введение большого числа различных режимов адресации;

· обладает сложной кодировкой инструкции.

Процессору с архитектурой CISC приходится иметь дело с более сложными инструкциями неодинаковой длины. Выполнение одиночной CISC-инструкции может происходить быстрее, однако обрабатывать несколько таких инструкций параллельно сложнее.

RISC:

Процессор с сокращенным набором команд. Система команд имеет упрощенный вид. Все команды одинакового формата с простой кодировкой. Обращение к памяти происходит посредством команд загрузки и записи, остальные команды типа регистр-регистр. Команда, поступающая в CPU, уже разделена по полям и не требует дополнительной дешифрации. Отладка программ на RISC более сложна.

(Источник: викиверситет)

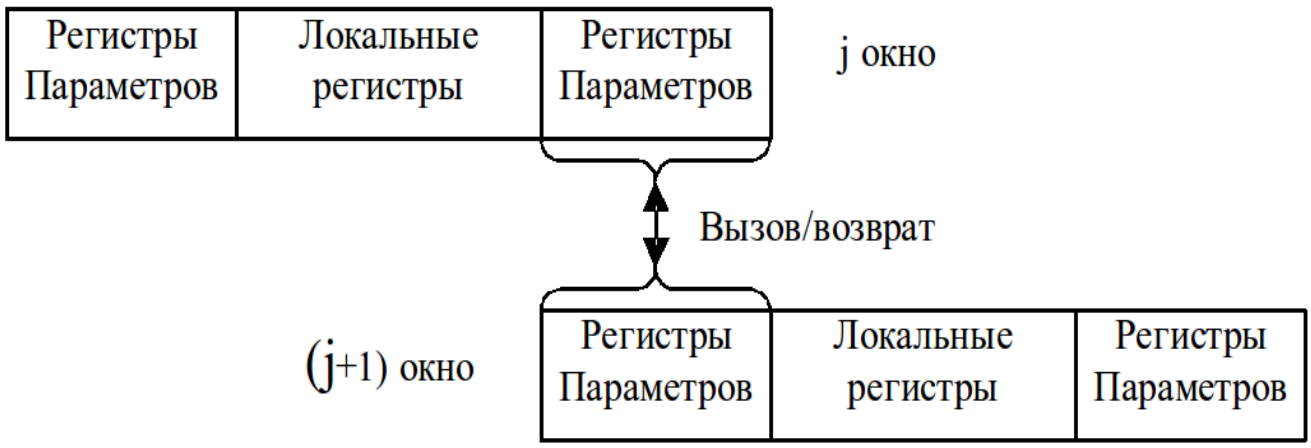
Из-за того, что команды в киске имеют офигительный разброс по сложности исполнения, то они плохо конвейризуются.

13. ФУНКЦИИ РЕГИСТРОВОГО ОКНА В RISC-ПРОЦЕССОРАХ.

ОТВЕТ:

Аппаратная оптимизация использования регистров сводится к сокращению затрат времени на работу с процедурой. Чтобы упростить и ускорить передачу параметров от вызывающей процедуры к вызываемой и обратно, было предложено использовать **регистровые окна**. Для этого все множество регистров(регистровый файл) разбивается на подмножества, называемые регистровыми окнами, и каждому окну соответствует одна процедура. В каждый момент времени только одно окно доступно и адресуемо. Все окна имеют один и тот же размер и состоят трех полей (областей фиксированного размера):

1. **Левое поле** окна включает регистры, которые содержат параметры процедуры, которая вызвала текущую процедуру, и результаты, которые необходимо передать обратно.
2. **Среднее поле** представляет собой локальные регистры, которые используются для хранения локальных переменных, как определено компилятором, и констант процедуры.
3. **Правое поле** (временные регистры) используются для обмена параметрами и результатами с процедурой, вызванной текущей процедурой.



Источник: лекции. Тут чуть-чуть понятно.

14. СРАВНИТЬ РЕАЛИЗАЦИЮ КОНВЕЙЕРА В VLIW И СУПЕРСКАЛЯРНЫХ ПРОЦЕССОРАХ .

ОТВЕТ: в общем, в суперскалярах конвейер сложный, потому что ему нужно подбирать команды, которые не пересекаются по различным параметрам, типа данных или ресурсов, выкидывать команды, которые оказались неверно предсказанными и т.п. VLIW ничем таким не занимается, за него всё это делает компилятор, он формирует последовательность из связок независимых команд, по длине равных ширине конвейера. Если компилятор не нашёл достаточное количество независимых команд, то он просто забивает связку NOP-ами (“нет операции”), то есть компилятор выполняет всё планирование выполнения потока команд, что в суперскалярах делает сам процессор. Это упрощает логику работы процессора, в результате на кристалле больше места для ресурсов и из-за этого ширина конвейера типичных VLIW - процессоров (6 - 8 команд) больше ширины конвейера типичных

суперскалярных процессоров (3 - 5 команд). Совсем напрямую: в суперскаляре конвейер умный и медленный, в VLIW конвейер тупой и быстрый.

(Источник: методичка стр. 60, тут какие-то ошметки информации есть)

15. СРАВНИТЬ VLIW-ПРОЦЕССОР И СУПЕРСКАЛЯРНЫЙ ПРОЦЕССОР С ТОЧКИ ЗРЕНИЯ ВЫЯВЛЕНИЯ И РЕАЛИЗАЦИИ ILP.

ОТВЕТ:

ILP-процессоры - процессоры с параллелизмом на уровне команд.

Реальная производительность ILP-процессора на конкретной программе зависит от количества независимых команд, которые могут быть выполнены параллельно, и ограничивается шириной конвейера.

По способу выявления независимых команд компьютеры данного класса различаются суперскалярные процессоры и VLIW-процессоры (Very Long Instruction Word), т.е. процессоры с длинным командным словом.

	Суперскалярный	**VLIW**
Чем выявляется ILP?	Диспетчером в процессоре	Компилятором
Как выявляется ILP?	Множество очередных последовательно идущих и ожидающих выполнения команд программы образуют так называемое «**окно просмотра**». В рамках этого окна диспетчер ищет готовые к выполнению команды, по несколько за такт, и отправляет их на выполнение к соответствующим	Компилятор для VLIW-процессора выполняет статический анализ программы, выявляет независимые операции и формирует последовательность связок для выполнения VLIW-процессором. Число команд в связке определяется архитектурой процессора и равно ширине

	функциональным устройствам. Команда считается готовой к выполнению, если все ее зависимости разрешены.	конвейера. Для каждой простой команды в коде указаны все необходимые аппаратные ресурсы для ее выполнения: аппаратные регистры, функциональные устройства. Если компилятор не нашел достаточное количество независимых команд для формирования связки, он дополняет ее командами NOP («нет операции»). По сути, компилятор выполняет почти полное детальное планирование выполнения потока команд на VLIW-процессоре.
Как команды исполняются?	В процессоре с **упорядоченным выполнением команд** диспетчер рассматривает команды только в том порядке, в котором они идут во входной последовательности. В процессоре с **выполнением команд вне порядка** (OoO) диспетчер рассматривает команды в пределах всего окна просмотра. Для повышения производительности	Диспетчер VLIW-процессора только выполняет закодированные в программе указания, т.к все уже проанализировано и распланировано компилятором.

	суперскалярный процессор вынужден разрешать ложные зависимости с помощью механизма **переименования регистров**.	

Суперскалярный процессор:

Плюсы:

1. использование для распараллеливания информации, которая доступна только в момент выполнения программы;
2. Производительность суперскалярных процессоров, в отличие от VLIW-процессоров, в меньшей степени зависит от качества кода, что обеспечивает хорошую переносимость программ и хорошую производительность «в среднем»

Минусы:

1. Высокая сложность аппаратного обеспечения;
2. На кристалле мало места под ресурсы;
3. В динамике не может выявить все независимые команды в программе.

VLIW-процессор:

Плюсы:

1. Простая логика работы процессора по сравнению с суперскалярными процессорами;
2. На кристалле много места под ресурсы;
3. Ширина конвейера типичных VLIW-процессоров (6-8 команд) больше ширины конвейера типичных суперскалярных процессоров;
4. Производительнее суперскалярных.

Минусы:

1. Часть параллелизма не может быть выявлена вообще, поскольку у компилятора нет информации о зависимостях, которые формируются в процессе вычисления;
2. Производительность VLIW-процессоров сильно зависит от качества кода, а VLIW-код жестко «привязан» к конкретной микроархитектуре процессора.

Источник: методичка стр. 56 - 61.

16. СРАВНИТЬ СТАТИЧЕСКИЙ И ДИНАМИЧЕСКИЙ ПРЕДСКАЗАТЕЛИ ПЕРЕХОДОВ.

Динамическое предсказание перехода - предсказание о наиболее вероятном исходе команды условного перехода, которое принимается, исходя из собираемой в процессе выполнения программы информации о предшествующих переходах (методичка стр. 155)

Статическое предсказание перехода - предсказание наиболее вероятного результата команды условного перехода, закодированное в коде команды. Статическое предсказание основывается на некотором априорном знании компилятора о ходе выполнения программы. (методичка стр. 162)

Тут говорится, что статическое предсказание в наше время используется только тогда, когда динамическое использовать невозможно. Видимо, это те случаи, когда нельзя собрать статистику исполнения программы - она мала или очень неоднородная и динамическое предсказание оказывается мало- или вообще не-

эффективным. Статические предсказания это примитивные штуки, например в современных процах реализовано такое предсказание - любой переход назад (на младшие адреса) будет выполнен и если такой переход встречается, то в конвейер загружаются инструкции, расположенные по адресу перехода. А любой переход вперед (на старшие адреса) предположительно, выполнен не будет а загружаются инструкции идущие после инструкции перехода. Такой метод используется в качестве “подстраховки” и очевидно, что динамический метод, если его можно применить, будет более эффективен. Но если например в программе всего один переход, то динамический метод не сработает вообще, а статический с вероятностью 50% предскажет то, что нужно.

17. РЕКОМЕНДАЦИИ ЭФФЕКТИВНОГО ПРОГРАММИРОВАНИЯ С УЧЕТОМ ОРГАНИЗАЦИИ ПАМЯТИ.

ПРОБЛЕМЫ И СПОСОБЫ ИХ РЕШЕНИЯ:

1. **Данные на границе блоков памяти**(первые несколько байт переменной могут располагаться в одном блоке, а несколько – в следующем)
    a. **Выравнивание данных** – это изменение положения переменных в памяти таким образом, чтобы они были выровнены относительно некоторой величины(размера переменной или блока):
        i. Может производиться компилятором (автоматически или с ключами);
        ii. При динамическом выделении памяти можно пользоваться _mm_malloc или posix_memalign для выравнивания;

iii. Можно также выделить область памяти большего размера с помощью стандартной функции malloc и сделать в начале области необходимый для выравнивания отступ;

iv. Нужно осторожнее работать с указателями, так как можно получить невыровненные данные.

2. **Разреженное размещение данных в памяти**(данные«разбросаны» по памяти на значительные расстояния друг от друга (больше размера кэш-строки))
    a. Разрешить компилятору производить выравнивание данных (обычно он это делает по умолчанию);
    b. Избегать особых ситуаций явного сдвига адреса элемента на значение, не кратное размеру элемента (как в листинге 8);
    c. Группировать данные в соответствии с их использованием;
    d. Размещать данные в структурах как можно более плотно.
3. **Данные, смещенные на величину, кратную размеру банка кэш-памяти**
   Для того, чтобы избежать эффекта **«буксования»** кэш-памяти, обычно используются два приема:
    1. Изменение порядка обхода элементов;
    2. Изменение расстояния между элементами(добавить фиктивные элементы в массив(не меньше размера кэш-строки)).
2. **Нарушение локальности во времени обращений в память**(блок памяти понадобился, но уже вытеснен из кэша другими блоками)
    1. Для решения этой проблемы программу перестраивают таким образом, чтобы если некоторая переменная была загружена из оперативной памяти в кэш-память, то над ней выполняются все возможные вычисления, пока она еще находится в кэш-памяти.

3. **Неупорядоченный обход данных в памяти**

   Способы достижения последовательного обхода памяти в программе:

   1. Перестановка циклов;
   2. Изменение порядка следования размерностей массива(транспонировать матрицу).

Источник: методичка, стр. 99 - 115.

18. Сравнить способы реализации условного перехода в суперскалярных процессорах и VLIW-процессорах.
19. ПРОГРАММНЫЙ СПОСОБ КОНВЕЙЕРИЗАЦИИ ЦИКЛОВ.

ОТВЕТ:

Программная конвейеризация циклов - это техника, используемая компиляторами для оптимизации циклов по аналогии с вычислительным конвейером в микропроцессорах. Является формой внеочередного исполнения с той разницей, что переупорядочивание выполняется не процессором, а компилятором (либо, в случае ручной оптимизации, программистом). Некоторые компьютерные архитектуры, например Intel IA-64, имеют явную аппаратную поддержку для упрощения программной конвейеризации циклов. При конвейеризации цикла в каждый момент времени на исполнении находится код, относящийся к нескольким итерациям цикла, но к различным частям цикла.

Следующие аппаратные решения упрощают описанную оптимизацию:

1. **«Вращающиеся регистры»** - часть регистрового файла отводится на область вращающихся регистров. Инструкции, использующие некоторый архитектурный регистр из этой области, будут обращаться к различным физическим регистрам по мере исполнения итераций (и продвижения вращающейся области). Через какое-то количество итераций вновь произойдет

обращение к исходному физическому регистру. Это позволяет работать с различными итерациями цикла одновременно, и при этом не требуется явных пересылок между регистрами;

2. **Предикаты и предикатное исполнение инструкций**, при котором предикатом работает некоторые специальные предикаты цикла. Эти предикаты позволяют включать и отключать некоторые инструкции цикла в процессе прохождения итераций, тем самым реализуя пролог и эпилог цикла в основном его коде, а также упрощают раскрытие условных операций в теле цикла;
3. **Аппаратная поддержка циклов**, при которой программа дает процессору информацию о размере цикла и о параметрах индексной переменной. Это позволяет сократить накладные расходы на организацию цикла. Также позволяет настроить скорость вращения и размер группы вращающихся регистров.

Источник: Википедия, методичка по программной конвейеризации циклов на платформе ARM от ИСП РАН.

20. Оптимизирующий компилятор в RISC-процессорах.
21. ПОНЯТИЕ РЕГИСТРОВОГО ОКНА.

    ОТВЕТ: смотреть вопрос 13.
22. Средства повышения производительности в процессоре Itanium.
23. ОРГАНИЗАЦИЯ ВЗАИМОДЕЙСТВИЯ НЕСКОЛЬКИХ УСТРОЙСТВ НА ШИНЕ. РАЗДЕЛЕНИЕ НА ВЕДУЩИЕ И ВЕДОМЫЕ УСТРОЙСТВА.

    ОТВЕТ:

    Шина - это коммуникационное аппаратное обеспечение, представляющее собой набор проводников, несущих двоичные сигналы.

Шинный интерфейс – иерархия шин вместе с их протоколами работы, которые обеспечивают связь процессора с кэш памятью, оперативной памятью и периферийными устройствами.

Функции шин:

1. Передача данных и команд;
2. Организация взаимодействия устройств на шине;
3. Обнаружение и обработка ошибок в программном обеспечении или аппаратуре.

Когда два устройства обмениваются информацией по шине, одно из них должно инициировать обмен и управлять им. Такого рода устройства называют **ведущими** (bus master). В компьютерной терминологии «ведущий» — это любое устройство, способное взять на себя владение шиной и управлять пересылкой данных. Ведущий не обязательно использует данные сам. Он, например, может захватить управление шиной в интересах другого устройства. Устройства, не обладающие возможностями инициирования транзакции, носят название **ведомых** (bus slave). В принципе к шине может быть подключено несколько потенциальных ведущих, но в любой момент времени активным может быть только один из них: если несколько устройств передают информацию одновременно, их сигналы перекрываются и искажаются. Для предотвращения одновременной активности нескольких ведущих в любой шине предусматривается процедура допуска к управлению шиной только одного из претендентов **(арбитраж)**. В то же время некоторые шины допускают широковещательный режим записи, когда информация одного ведущего передается сразу нескольким ведомым (здесь арбитраж не требуется). Сигнал, направленный одним устройством, доступен всем остальным устройствам, подключенным к шине.

Источник: лекции, методичка СибГУТИ.

24. Переименование регистров.

25. СРАВНИТЬ ЦЕНТРАЛИЗОВАННЫЙ И ДЕЦЕНТРАЛИЗОВАННЫЙ СПОСОБЫ ОРГАНИЗАЦИИ ВЗАИМОДЕЙСТВИЯ УСТРОЙСТВ НА ШИНЕ.

ОТВЕТ:

Организация взаимодействия устройств на шине осуществляется механизмом арбитража.

Типы арбитража:

1. Централизованный;
2. Децентрализованный.

	Централизованный	**Децентрализованный**
Кем осуществляется?	Специальное устройство - **центральный арбитр**, - ответственное за предоставление доступа к шине только одному из запросивших ведущих.	Каждый ведущий содержит блок управления доступом к шине, и при совместном использовании шины такие блоки взаимодействуют друг с другом, разделяя между собой ответственность за доступ к шине.
Способ подключения ведущих устройств к арбитру?	**Параллельный**: центральный арбитр связан с каждым потенциальным ведущим индивидуальными двухпроводными трактами. Запросы к центральному арбитру могут поступать независимо и параллельно. **Последовательный**: это такой арбитраж, при	Нет, так как нет центрального арбитра.

	котором все ведущие устройства передают сигнал разрешения захвата шины по цепочке от более приоритетных устройств к менее приоритетным до тех пор, пока он не достигнет ведущего, посылавшего сигнал запроса на захват шины. Изначально сигнал формируется центральным арбитром.	
Отказоустойчивость?	Центральный арбитр - единственная точка отказа системы.	Система устойчива к отказам.

Источник: лекции, методичка СибГУТИ.

26. Классификация периферийных устройств, назначение и основные устройства.
27. МЕХАНИЗМ ПРЕРЫВАНИЙ. АРБИТРАЖ ШИН И СХЕМЫ АРБИТРАЖА.

ОТВЕТ:

Арбитраж шин и схемы арбитража - вопросы 25 и 23.

Прерывание (англ. interrupt) — сигнал от программного или аппаратного обеспечения, сообщающий процессору о наступлении какого-либо события, требующего немедленного внимания. Прерывание извещает процессор о наступлении высокоприоритетного события, требующего прерывания текущего кода, выполняемого процессором. Процессор отвечает приостановкой своей текущей активности, сохраняя свое состояние и выполняя функцию, называемую обработчиком прерывания (или программой обработки прерывания), которая

реагирует на событие и обслуживает его, после чего возвращает управление в прерванный код.

В зависимости от источника возникновения сигнала прерывания делятся на:

1. **Асинхронные, или внешние** (аппаратные) — события, которые исходят от внешних аппаратных устройств (например, периферийных устройств) и могут произойти в любой произвольный момент: сигнал от таймера, сетевой карты или дискового накопителя, нажатие клавиш клавиатуры, движение мыши. Факт возникновения в системе такого прерывания трактуется как запрос на прерывание (англ. Interrupt request, IRQ) - устройства сообщают, что они требуют внимания со стороны ОС;
2. **Синхронные, или внутренние** — события в самом процессоре как результат нарушения каких-то условий при исполнении машинного кода: деление на ноль или переполнение стека, обращение к недопустимым адресам памяти или недопустимый код операции;
3. **Программные** (частный случай внутреннего прерывания) — инициируются исполнением специальной инструкции в коде программы. Программные прерывания, как правило, используются для обращения к функциям встроенного программного обеспечения (firmware), драйверов и операционной системы.

До окончания обработки прерывания обычно устанавливается запрет на обработку этого типа прерывания, чтобы процессор не входил в цикл обработки одного прерывания. Приоритизация означает, что все источники прерываний делятся на классы и каждому классу назначается свой уровень приоритета запроса на прерывание.

1. **Относительное обслуживание прерываний** означает, что если во время обработки прерывания поступает более приоритетное прерывание, то это прерывание будет обработано только после завершения текущей процедуры обработки прерывания;
2. **Абсолютное обслуживание прерываний** означает, что если во время обработки прерывания поступает более приоритетное прерывание, то текущая процедура обработки прерывания вытесняется, и процессор начинает выполнять обработку вновь поступившего более приоритетного прерывания. После завершения этой процедуры процессор возвращается к выполнению вытесненной процедуры обработки прерывания.

Источник: Википедия.

28. Общие черты у RISC-процессоров и VLIW-процессоров.
29. ВИДЫ ШИН В ЭВМ, ИХ НАЗНАЧЕНИЕ И ОСНОВНЫЕ ХАРАКТЕРИСТИКИ.

ОТВЕТ:

Виды шин:

1. **Шина «процессор-память»:**

   Шина «процессор-память» обеспечивает непосредственную связь между центральным процессором (ЦП) вычислительной машины и основной памятью (ОП). В современных микропроцессорах такую шину часто называют шиной переднего плана и обозначают аббревиатурой FSB (Front-Side Bus). Интенсивный трафик между процессором и памятью требует, чтобы полоса пропускания шины, то есть количество информации, проходящей по шине в единицу времени, была наибольшей. Роль этой шины иногда выполняет системная шина (см. ниже), однако в плане эффективности значительно выгоднее, если обмен между ЦП и ОП ведется по отдельной шине. К

рассматриваемому виду можно отнести также шину, связывающую процессор с кэш-памятью второго уровня, известную как шина заднего плана — BSB (Back-Side Bus). BSB позволяет вести обмен с большей скоростью, чем FSB, и полностью реализовать возможности более скоростной кэш-памяти.

2. **Шины ввода/вывода:**

   Шина ввода/вывода служит для соединения процессора (памяти) с устройствами ввода/вывода (УВВ). Учитывая разнообразие таких устройств, шины ввода/вывода унифицируются и стандартизируются. Связи с большинством УВВ (но не с видеосистемами) не требуют от шины высокой пропускной способности. При проектировании шин ввода/вывода в учет берутся стоимость конструктивна и соединительных разъемов. Такие шины содержат меньше линий по сравнению с вариантом «процессор-память», но длина линий может быть весьма большой. Типичными примерами подобных шин могут служить шины PCI и SCSI.

3. **Системные шины:**

   Системная шина служит для физического и логического объединения всех устройств ВМ. Поскольку основные устройства машины, как правило, размещаются на общей монтажной плате, системную шину часто называют объединительной шиной (backplane bus), хотя эти термины нельзя считать строго эквивалентными.

   Системная шина в состоянии содержать несколько сотен линий. Совокупность линий шины можно подразделить на три функциональные группы:

   1. **шина данных**;
   2. **шина адреса**;

3. **шина управления**;

Основные характеристики шин:

1. **Разрядность шины** определяется числом параллельных проводников, входящих в нее;
2. **Тактовая частота**;
3. **Пропускная способность** шины определяется количеством байт информации, передающейся в шине за секунду. Для определения пропускной способности шины необходимо умножить тактовую частоту шины на её разрядность.

Источник: методичка СибГУТИ.

30. Уровни программирования периферийных устройств. Примеры сред программирования.
31. ОБЩИЕ И РАЗНЫЕ ЧЕРТЫ КОНВЕЙЕРОВ ПРОЦЕССОРОВ POWER4 И ITANIUM?

Update 2018:

Вопросы к зачету по курсу «ЭВМ и периферийные устройства»

1. ОПРЕДЕЛЕНИЕ АРХИТЕКТУРЫ И МИКРОАРХИТЕКТУРЫ КОМПЬЮТЕРА. АРХИТЕКТУРНЫЕ ПРИНЦИПЫ КОМПЬЮТЕРА ФОН НЕЙМАНА.

   ОТВЕТ:

Архитектура компьютера - логическое представление компьютера с точки зрения программиста.

Каждая архитектура может иметь несколько аппаратных реализаций.

Аппаратная реализация компьютера называется организацией или **микроархитектурой**. Микроархитектура определяет структуру компьютера, а именно, набор компонентов компьютера, их связи, функциональные возможности каждого компонента (например, количество арифметических

Логических устройств, число стадий конвейера, размер аппаратного регистрового файла или разрядность шины между оперативной памятью и процессором).

Принципы компьютера Фон Неймана:

a. **Принцип программного управления.** Алгоритм решения задачи должен быть представлен в виде программы, состоящей из последовательности команд. Каждая команда должна принадлежать некоторому набору команд, реализуемых компьютером. Команды выполняются последовательно. Именно эта последовательность команд и управляет работой компьютера;
b. **Принцип хранимой программы.** Команды представляются в числовой форме и хранятся в оперативной памяти вместе с данными, а не задаются аппаратными средствами. Кроме того, в отличие от раздельного хранения команд и данных, это позволяет экономно распределить память между командами и данными;
c. **Синхронное функционирование в ритме, задаваемом тактовым генератором.** Команды выполняются последовательно каждая за определенный квант времени, называемый тактом. Продолжительность такта фиксирована и определяется частотой тактового генератора, или тактовой частотой;

d. **Принцип условного перехода.** В наборе команд компьютера имеются специальные команды условных переходов. В зависимости от своего операнда, команда условного перехода может выбрать инструкцию в программе, которую требуется выполнять далее. Таким образом, возможно, организовывать циклы, итерационные процессы и т. д.

e. **Принцип использования двоичной системы счисления для представления информации.** Наиболее технологичной для аппаратной реализации оказалась двоичная система счисления, поэтому в компьютере все числа хранятся и обрабатываются именно в этой системе счисления. К логическим схемам, построенным для этой системы счисления, может быть применен аппарат булевой алгебры.

f. **Принцип иерархичности запоминающих устройств.** Причиной его введения стало несоответствие в стоимости и быстродействии различных типов памяти. Этот принцип предписывает располагать программы и данные для долговременного хранения на дешевой медленной памяти большого объема, а программы и данные, используемые в процессе вычислений, на дорогой быстрой памяти малого объема.

Источник: методичка, стр 3 - 7.

2. ОСНОВНЫЕ КЛАССЫ АРХИТЕКТУР.

   ОТВЕТ: CISC, RISC, VLIW (наследник RISC. Itanium и Эльбрус это VLIW) CISC, RISC, VLIW

   CISC:

- нефиксированное значение длины команды
- арифметические действия кодируются в одной команде;
- небольшое число регистров, каждый из которых выполняет строго определённую функцию.

RISC:

- Фиксированная длина машинных инструкций (например, 32 бита) и простой формат команды. *В киске команды разной длины и это хуже для конвейера, так как команды выполняются за различное число тактов и конвейер будет стоять и ждать, пока выполнится какая-нибудь сложная команда, в риске такой лажи нет.*
- Специализированные команды для операций с памятью — чтения или записи. Операции вида Read-Modify-Write («прочитать-изменить-записать») отсутствуют. Любые операции «изменить» выполняются только над содержимым регистров (т. н. архитектура load-and-store). *Простыми словами, тут сначала загружаешь в регистр, преобразовываешь и затем выгружаешь в память, в киске всё это можно сделать одной командой например.*
- Большое количество регистров общего назначения (32 и более).
- Отсутствие поддержки операций вида «изменить» над укороченными типами данных — байт, 16-разрядное слово. *Например, если регистр 64 бита, то в риске нельзя изменить первые 16 бит, или первые 32 бита, только через танцы с бубном. В киске это можно, там в зависимости от первой буквы в имени регистра (типа RAX, EAX) мы меняем те или иные биты.*
- Отсутствие микропрограмм внутри самого процессора. То, что в CISC-процессоре исполняется микропрограммами, в RISC-процессоре исполняется как обыкновенный (хотя и помещенный в специальное

хранилище) машинный код, не отличающийся принципиально от кода ядра ОС и приложений.

VLIW:

Характеризуется тем, что одна инструкция процессора содержит несколько операций, которые должны выполняться параллельно.

В суперскалярных процессорах также есть несколько вычислительных модулей, но задача распределения работы между ними решается аппаратно. Это сильно усложняет устройство процессора, и может быть чревато ошибками. В процессорах VLIW задача распределения решается во время компиляции и в инструкциях явно указано, какое вычислительное устройство какую команду должно выполнять. Т.е. от аппаратного усложнения ушли к программному.

VLIW можно считать логическим продолжением идеологии RISC, расширяющей её на архитектуры с несколькими вычислительными модулями. Так же, как в RISC, в инструкции явно указывается, что именно должен делать каждый модуль процессора. Из-за этого длина инструкции может достигать 128 или даже 256 бит.

Подход VLIW сильно упрощает архитектуру процессора, перекладывая задачу распределения вычислительных устройств на компилятор.

Из-за большого количества пустых инструкций для простаивающих устройств программы для VLIW-процессоров могут быть гораздо длиннее, чем аналогичные программы для традиционных архитектур.

Программирование вручную, на уровне машинных кодов для VLIW-архитектур, является достаточно сложным. Приходится полагаться на оптимизацию компилятора.

3. ТРАДИЦИОННАЯ АРХИТЕКТУРА ФОН НЕЙМАНА. ОСНОВНЫЕ АРХИТЕКТУРНЫЕ ПРИНЦИПЫ ПОСТРОЕНИЯ КОМПЬЮТЕРА (ЭВМ). ОГРАНИЧЕНИЯ АРХИТЕКТУРЫ ФОН НЕЙМАНА.

Компьютер фон Неймана включает в себя 4 основных компонента: *оперативка, устройство управления, арифметико-логическое устройство, устройство вводы-вывода и внешняя память.*

Основные принципы:

- **Принцип программного управления.** Алгоритм должен быть представлен в виде программы из последовательности команд, выполняющихся последовательно.
- **Принцип хранимой программы.** Команды представляются в числовой форме и хранятся в оперативке вместе с данными, а не задаются аппаратно.
- **Синхронное функционирование в ритме, задаваемом тактовым генератором.** Команды выполняются последовательно, каждая за определенный квант времени, называемый тактом. Продолжительность такта фиксирована и определяется частотой тактового генератора или тактовой частотой.
- **Принцип условного перехода.** В наборе команд компьютера имеются специальные команды условных переходов. В зависимости от своего операнда, команда условного перехода может выбрать инструкцию в программе, которую требуется выполнять далее. Таким образом, можно делать циклы, итерации и т.д.
- **Принцип использования двоичной системы счисления для представления информации**. Наиболее технологичной для аппаратной реализации оказалась двоичная система счисления, поэтому в компьютере все числа хранятся и обрабатываются именно в этой системе счисления. К логическим схемам, построенным для этой системы счисления, может быть применен аппарат булевой алгебры.
- **Принцип иерархичности запоминающих устройств.** Причиной его введения стало несоответствие в стоимости и быстродействии различных типов памяти. Этот принцип предписывает располагать программы и данные для долговременного хранения н дешевой медленной памяти большого объема, а программы и данные, используемые в процессе вычислений, на дорогой быстрой памяти малого объема.

***Ограничения**:* совместное хранение данных и программы и последовательное выполнения команд привело к так называемому “бутылочному горлу”: доступ к командам и данным осуществляется через один

и тот же канал. Пропускная способность этого канала ограничивает скорость работы компьютера.

Источник: методичка стр 5

4. АРХИТЕКТУРНЫЕ УСОВЕРШЕНСТВОВАНИЯ КОМПЬЮТЕРА

Оптимизация подсистемы памяти:

- Контроллер памяти
- Высокоскоростная шина
- Кэш и иерархия памяти
- Виртуальная память
- Аппаратная предвыборка данных и команд

Оптимизация выполнения команд:

- Конвейеризация
- Упрощение набора команд
- Истинный параллелизм
- Данные (*что за данные только Марковой известно*)
- Инструкции
- Потоки
- Программы (*что она хотела этим сказать, наверное, даже ей не известно)*

Источник: презентация Марковой
ssd.sscc.ru/old/chair/files/architecture/archittecture2010.ppt

5. УПРАВЛЯЮЩИЕ СТРАТЕГИИ КОМПЬЮТЕРОВ.

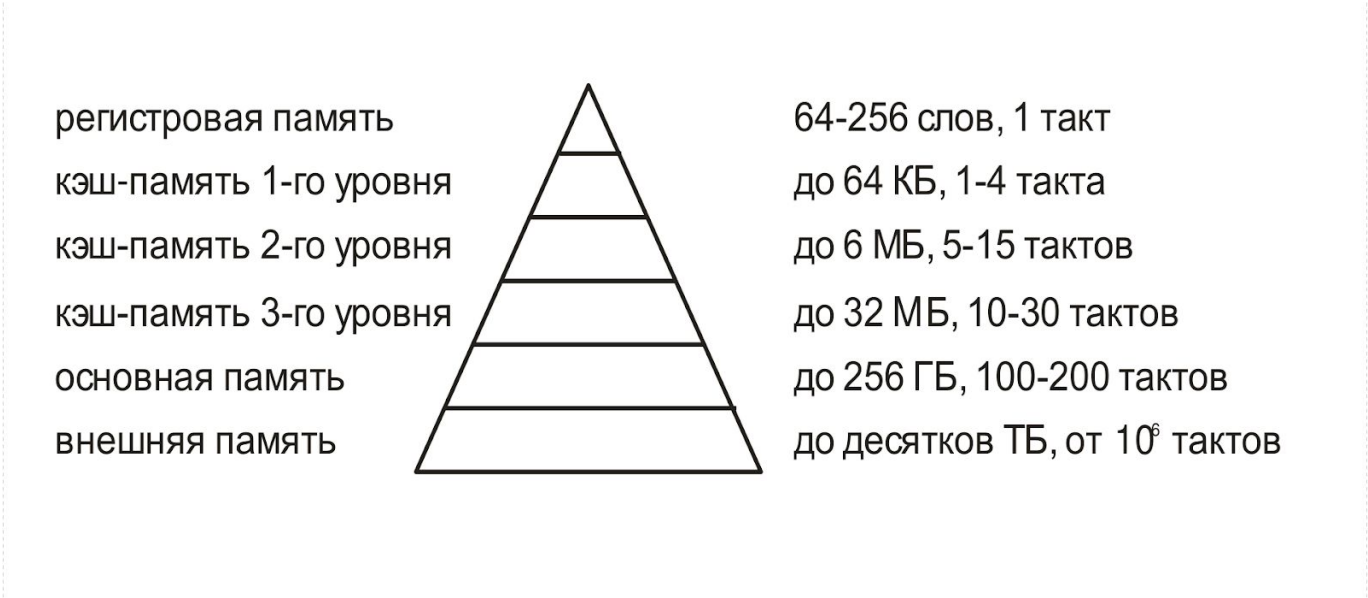
ОТВЕТ:

1. Команда выполняется, если предыдущая команда, определенная в машинном коде, выполнена (control flow).
2. Команда выполняется, когда требуемые операнды готовы(data flow).

3. Команда выполняется, когда ее результат требуется другой команде (demand driven).
4. Команда выполняется, когда появляются частичные образы данных(pattern driven).

6. ОРГАНИЗАЦИЯ ПАМЯТИ (ОСНОВНЫЕ ПОНЯТИЯ: АДРЕС, ЯЧЕЙКА, СЛОВО, РЕГИСТР, РЕГИСТРОВЫЙ ФАЙЛ, КОМАНДА, ПАМЯТЬ)

Устройство памяти:



Адрес - уникальный номер ячейки памяти.

Ячейка - минимально адресуемая единица информации. Компьютер может прочитать или записать значение в ячейку только целиком. Практически во всех современных компьютерах размер ячейки равен одному байту.

Слово - единица хранения информации, число разрядов в которой определяется разрядностью архитектуры. Например, для 32-разрядной архитектуры размер слова равен 32 бита или 4 байта.

Регистр - есть программный регистр как часть программной архитектуры или аппаратный регистр как часть аппаратной архитектуры. Отображение программных на аппаратные выполняется с помощью механизма переименования регистров.

Программный регистр - ячейка памяти, часть программной архитектуры компьютера. Доступ к программному регистру осуществляется по имени.

Аппаратный регистр - ячейка памяти, расположенная на кристалле процессора. В отличие от других типов запоминающих устройств регистры являются самыми быстрыми.

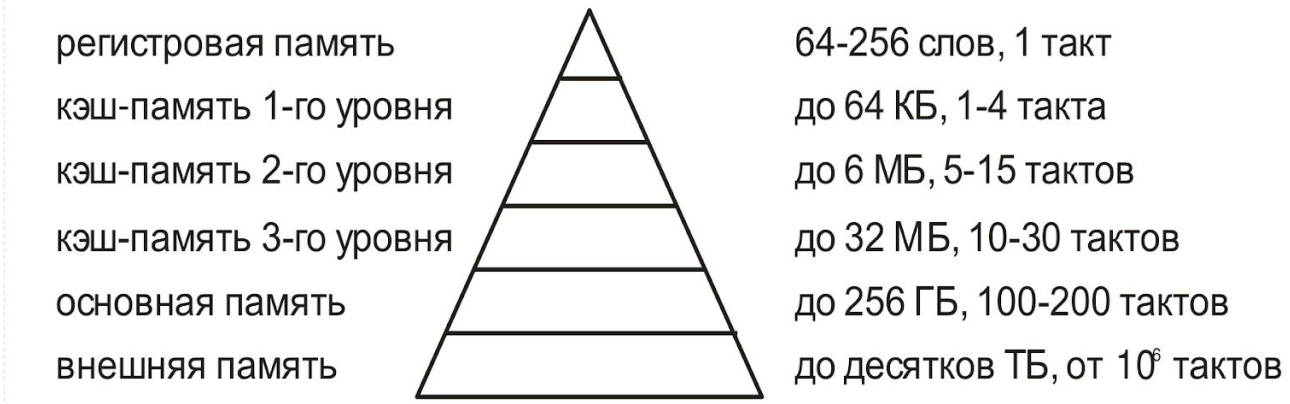
Регистровый файл - функциональный блок процессора, который содержит аппаратные регистры.

Команда - элементарная операция, выполняемая процессором.

Память - в аппаратной архитектуре общее название различных типов запоминающих устройств, в программной архитектуре общее название для ячеек, способных хранить числовые значения.

Источник: словарик в методичке (в конце)

7. Иерархия памяти. Требования к расположению уровней иерархии.



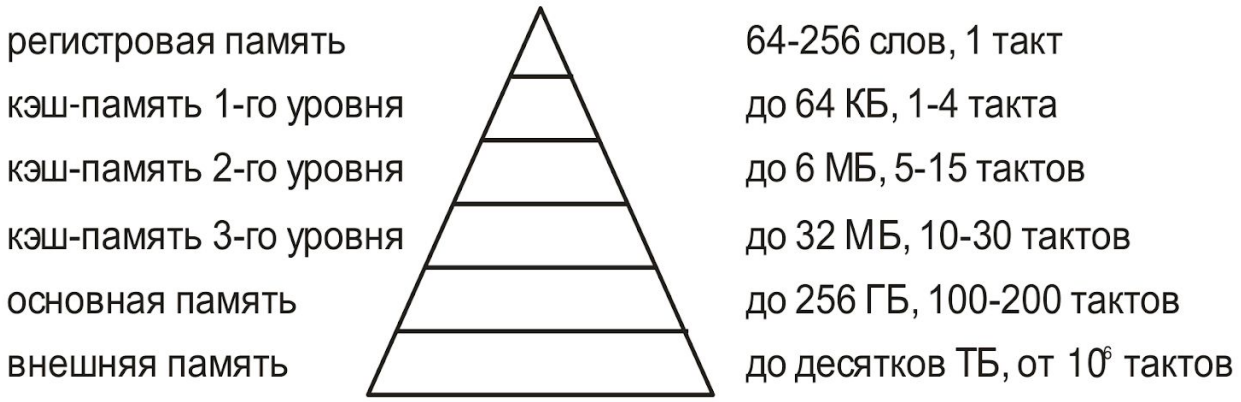
Требования:

С увеличением уровня иерархии должно происходить:

- Уменьшение стоимости хранения единицы данных на данном уровне
- Увеличения объема памяти данного уровня
- Увеличение времени доступа
- Уменьшение частоты обращений к уровню со стороны процессора

Источник: методичка стр. 15

8. ТИПИЧНАЯ СХЕМА ИЕРАРХИИ ПАМЯТИ.



9. КЭШ-ПАМЯТЬ. ПРИНЦИПЫ ОРГАНИЗАЦИИ КЭШ-ПАМЯТИ. ЗА СЧЕТ ЧЕГО ПОЛУЧАЕТСЯ ВЫИГРЫШ ВО ВРЕМЕНИ?

Кэш-память это высокоскоростная память небольшого размера с прямым доступом. Она предназначена для временного хранения фрагментов кода и данных. Кэш-память охватывает все адресное пространство памяти, но в отличие от оперативной памяти, она не адресуема и невидима для программиста.

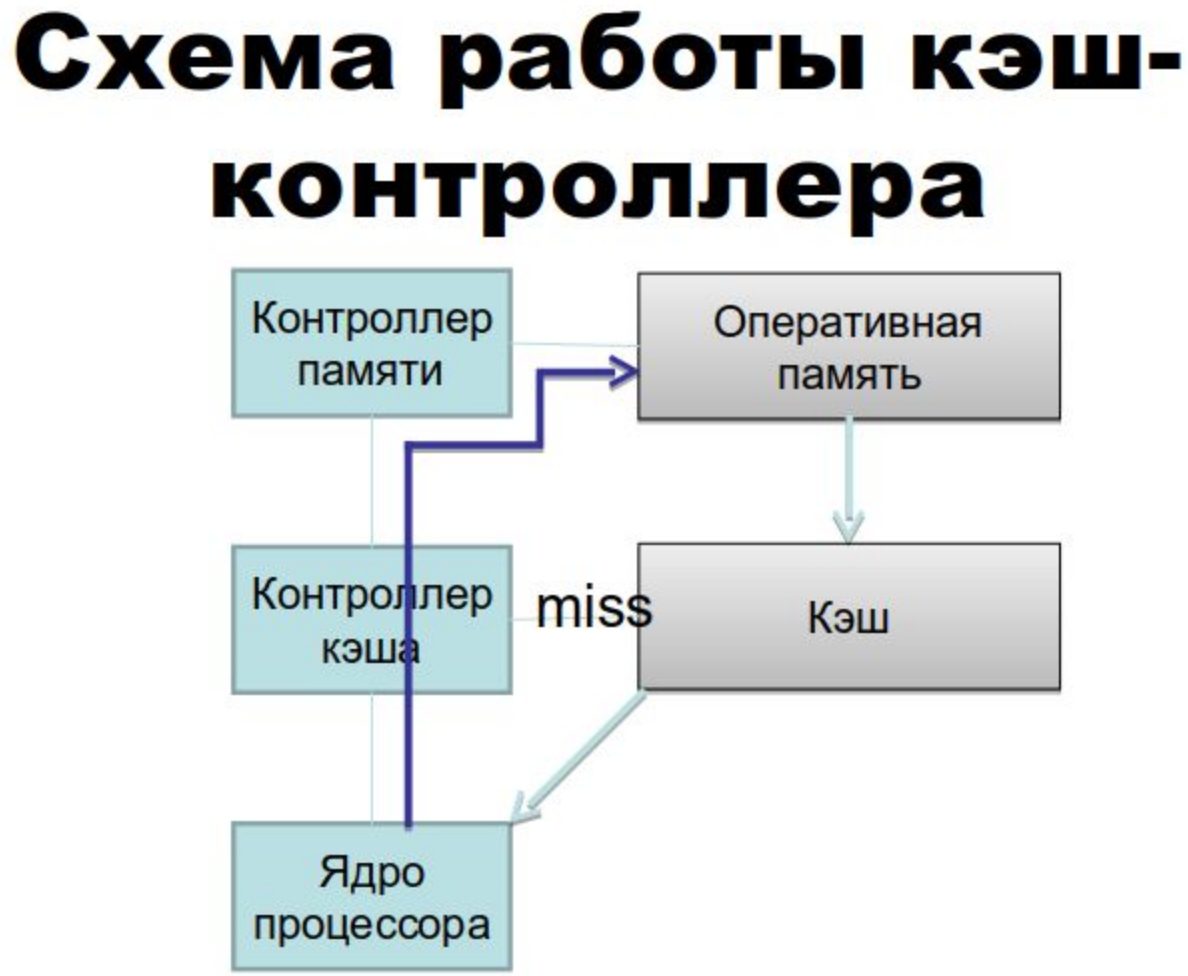
Выигрыш достигается засчёт предвыборки данных, а также потому что кэш быстрее обычной памяти. Кэш-контроллер предугадывает, какие данные понадобятся процессору и когда они понадобятся, то уже будут лежать в кэше, до которого процессору идти быстрее, чем до оперативки.

Источник: https://pandia.ru/text/78/353/127.php

10. СХЕМА РАБОТЫ КЭШ-КОНТРОЛЛЕРА

Если данные есть в кэше, то есть кэш-промах не случился, то достаём данные из кэша и отдаём процессору

Если промах, то идём в контроллер оперативки, потом из оперативки берем значение, кладём в кэш и оттуда в проц.



Источник: презентация
ssd.sscc.ru/sites/default/files/content/attach/374/lecture02.ppt

11. СРАВНИТЕЛЬНЫЙ АНАЛИЗ АЛГОРИТМОВ ОТОБРАЖЕНИЯ ДАННЫХ В КЭШ-ПАМЯТЬ.

ОТВЕТ:

Методичка стр. 27.

Существуют три схемы отображения адресов из оперативки в кэш: ***прямая***, ***ассоциативная*** и ***множественно-ассоциативная.***

a. **Прямое отображение**: адрес ячейки в оперативке состоит из тэга, индекса и смещения внутри блока. Тэг - старшая часть адреса, индекс - средняя, смещение - младшая. Оперативка отображается в кэш по правилу S = A mod C, где S - адрес строки кэша, A - адрес в оперативке, C - число строк в кэше. То есть каждая C-ая ячейка оперативки претендует на одну и ту же ячейку в одной и той же строке кэша. Таким образом, в строке кэша хранится тег той ячейки, которая в данный момент решила заехать к кэш и собственно данные этой ячейки. Младшая часть адреса ячейки оперативной памяти это смещение, которое показывает положение байта в строке кэша, так как в строке может помещаться больше одного байта (а может и нет, но это очень убогий кэш какой-то) (см. Картинку в методичке на стр. 28) Если на занятую ячейку претендует другая память, то возникнет буксование, потому что нужно вытеснить предыдущий элемент из кэша и обратиться в оперативку за новым.
b. **Ассоциативное отображение**: любой блок памяти может быть записан в любую строку кэша, то есть, тут буксования нет, разве что при полном заполнении кэша. При ассоциативном отображении адрес байта в оперативке представляется как тег и смещение внутри кэш-строки, то есть нет индекса, который определял в прямом отображении номер строки. Так как теперь строки кэша не пронумерованы, то для того, чтобы понять, лежат ли данные в кэше, нужно сравнить тег ячейки из оперативки со всеми тегами строк кэша. Это занимает больше времени, чем при прямом отображении, где нам сразу известно, в какой строке наши данные нужно искать. Таким образом,

избавились от буксования, но замедлили работу кэша вцелом. (методичка стр.29)

c. **Множественно-ассоциативное отображение**: суть в том, что теперь по одному индексу существуют несколько строк кэша. Обычно их количество это степени двойки 2, 4, 8, 16, но легко встретить и 12 например. Грубо говоря, кэш как бы разбивается на несколько кэшей прямого отображения. Один слой такого кэша это *банк*, количество банков - это *степень ассоциативности* кэша. Адрес состоит так же, как в прямом отображении, из тега, номера строки и смещения. Решение, в какой банк попадут данные, принимает кэш-контроллер, а затем уже в пределах одного банка всё работает как при прямом отображении. Буксование возникнуть может, но для этого нужно заполнить ячейку во всех банках, что случается, в общем случае, случается реже, чем заполнение одной ячейки в прямом отображении.

12. АЛГОРИТМЫ СОГЛАСОВАНИЯ СОДЕРЖИМОГО КЭШ-ПАМЯТИ И ОСНОВНОЙ ПАМЯТИ.

1. Когерентность с использованием справочника. Информация о состоянии блока физической памяти содержится только в одном месте, называемом справочником (физически справочник может быть распределен по узлам системы).

Протокол на основе “полного справочника” является наиболее простым из протоколов на основе директорий. Здесь присутствует единый централизованный справочник, хранящийся в основной памяти, который поддерживает информацию обо всех кэшах. Справочник содержит множество записей, описывающих каждую кэшируемую ячейку ОП. Обращение к справочнику производится каждый раз, когда один из процессоров изменяет

копию такой ячейки в своей локальной памяти. В этом случае информация из справочника нужна для того, чтобы аннулировать или обновить копии измененной ячейки (или всей строки, содержащей эту ячейку) в других локальных кэшах, где присутствует эта же строка. Для каждого блока ОП в справочнике выделяется одна запись, хранящая указатели на копии данной строки. Кроме того, в каждой записи выделен один бит модификации (D), показывающий, является ли копия “грязной”- (D = 1 - dirty) или “чистой”-(D = 0 - clean), то есть изменялось ли содержимое строки в кэш-памяти после того, как она была туда загружена. Этот бит указывает, имеет ли право процессор производить запись в данную строку.

2. Когерентность с использованием отслеживания. Каждый кэш, который содержит копию данных некоторого блока физической памяти, имеет также соответствующую копию служебной информации о его состоянии. Централизованная система записей отсутствует. Обычно кэши расположены на общей (разделяемой) шине и контроллеры всех кэшей наблюдают за шиной (просматривают ее) для определения того, не содержат ли они копию соответствующего блока.

Протокол **MESI:**

Протокол MESI реализован в современных многоядерных процессорах Intel и архитектуре PowerPC. Каждая строка кэша, согласно протоколу MESI, может находиться в одном из **четырех состояний**:

Modified - строка в кэше была изменена (отлична от основной памяти) и новое значение доступно только в данном кэше;

Exclusive - значение строки совпадает со значением в основной памяти, но его нет ни в одном другом кэше;

Shared - значение строки совпадает со значением в основной памяти и может присутствовать в других кэшах;

Invalid - строка кэша содержит недостоверную информацию.

3. Перехват. Когда из какого-либо одного кэша данные переписываются в оперативную память, контроллеры остальных получают сигнал об этом изменении ("перехватывают" информацию об изменении данных) и, если необходимо, изменяют соответствующие данные в своих кэшах.

Источники: Википедия, методичка Донецкого универа.

13. АЛГОРИТМЫ ЗАМЕЩЕНИЯ СТРОК КЭШ-ПАМЯТИ.

1. *Алгоритм замещения на основе наиболее давнего использования* (LRU - LEast Recently Used) Является наиболее эффективным алгоритмом замещения. В соответствии с этим алгоритмом замещается та строка кэша, к которой дольше всего не было обращения. Проводившиеся исследования показали, что алгоритм LRU работает достаточно хорошо в сравнении с оптимальным алгоритмом.
2. *Алгоритм замены наименее часто использовавшейся строки* (LFU - Least Frequently Used) В соответствии с этим алгоритмом заменяется та строка в кэше, к которой было меньше всего обращений. Аппаратная реализация алгоритма: каждая строка связывается со счетчиком обращений, к содержимому которого после каждого обращения добавляется единица. Главным претендентом на замещение является строка, счетчик которой содержит наименьшее число.
3. *Алгоритм, работающий по принципу FIFO* (первый вошёл, первый вышел) Заменяется строка, которая дольше всего находится в кэше, алгоритм легко реализуется с помощью рассмотренной очереди, с той лишь разницей, что после обращения к строке положение соответствующей ссылки в очереди не меняется.
4. *Произвольный выбор строки для замены.* Простейший алгоритм. Используется крайне редко.

Источник: https://studfiles.net/preview/3675581/page:18/

14.ЭФФЕКТИВНОЕ ПРОГРАММИРОВАНИЕ С УЧЕТОМ КЭШ-ПАМЯТИ.

1. Разреженное размещение данных в памяти(данные «разбросаны» по памяти на значительные расстояния друг от друга (больше размера кэш-строки))

a. Разрешить компилятору производить выравнивание данных (обычно он это делает по умолчанию);
b. Избегать особых ситуаций явного сдвига адреса элемента на значение, не кратное размеру элемента (как в листинге 8);
c. Группировать данные в соответствии с их использованием;
d. Размещать данные в структурах как можно более плотно.

2. Данные, смещенные на величину, кратную размеру банка кэш-памяти

Для того, чтобы избежать эффекта **«буксования»** кэш-памяти, обычно используются два приема:

e. Изменение порядка обхода элементов;
f. Изменение расстояния между элементами(добавить фиктивные элементы в массив(не меньше размера кэш-строки)).

3. Нарушение локальности во времени обращений в память(блок памяти понадобился, но уже вытеснен из кэша другими блоками)

g. Для решения этой проблемы программу перестраивают таким образом, чтобы если некоторая переменная была загружена из оперативной памяти в кэш-память, то над ней выполняются все возможные вычисления, пока она еще находится в кэш-памяти.

4. Неупорядоченный обход данных в памяти

Способы достижения последовательного обхода памяти в программе:

h. Перестановка циклов;
i. Изменение порядка следования размерностей массива(транспонировать матрицу).

Источник: методичка, стр. 99 - 115.

15. ВИРТУАЛЬНАЯ ПАМЯТЬ

Виртуальная память - это механизм управления иерархической памятью компьютера, который позволяет размещать в памяти и одновременно выполнять несколько процессов. Виртуальная память предполагает, что пользователи имеют дело с кажущейся одноуровневой памятью, объем которой равен всему адресному пространству.

Страничная виртуальная память - оперативка разбивается на страницы фиксированного размера. Сейчас наиболее используемый способ. Для ускорения доступа к таблицу преобразования адресов используются многоуровневые таблицы страниц и буферы быстрого преобразования адресов TLB.

Сегментная адресация памяти - оперативка делится на части произвольного размера - сегменты. Виртуальный адрес при сегментной организации памяти может быть представлен парой (g, s), где g — номер сегмента, а s — смещение в сегменте. Физический адрес получается путём сложения начального физического адреса сегмента, найденного в таблице сегментов по номеру g, и смещения s.

Недостатком данного метода распределения памяти является фрагментация на уровне сегментов и более медленное по сравнению со страничной организацией преобразование адреса.

Существует также гибридная странично-сегментная организация виртуальной памяти

Источник: методичка стр. 38, https://ru.wikipedia.org/wiki/Виртуальная_память

16. СПОСОБЫ УПРАВЛЕНИЯ ВИРТУАЛЬНОЙ ПАМЯТЬЮ. ИХ СРАВНЕНИЕ.

Способы управления виртуальной памятью

- страничный
- сегментный
- странично-сегментный

Страничный используется сейчас, значит он самый хороший. Сегментный позволяет делать сегменты разного размера и как бы раздавать им роли, но из-за неоднородности размеров, все действия с ними производятся медленнее.

Странично-сегментный разбивает память на сегменты, то каждый сегмент состоит из страниц фиксированного размера. То есть есть одна таблица сегментов и ещё у каждого сегмента своя таблица страниц.
Источник: презентация про виртуальную память

17. СПОСОБЫ ПРЕОБРАЗОВАНИЯ ВИРТУАЛЬНЫХ АДРЕСОВ В ФИЗИЧЕСКИЕ.

При страничном способе: виртуальный адрес представлен парой чисел (p, s) - где p - номер виртуальной страницы, а s - смещение внутри страницы. Физический адрес представляется аналогичной парой чисел (n, s). Если физическая страница в оперативке, то в таблице страниц отсчитывается p-я строка. Она содержит номер физической страницы n, по которому однозначно определяется физический адрес этой страницы. Искомый физический адрес вычисляется суммированием физического адреса физической страницы n и смещения s. Если страница не в оперативке, то ее нужно подгрузить в свободную страницу в оперативке, если таких нет, то какая-то страница вытесняется.

18. РЕКОМЕНДАЦИИ ЭФФЕКТИВНОГО ПРОГРАММИРОВАНИЯ С УЧЕТОМ ОРГАНИЗАЦИИ ПАМЯТИ.

ПРОБЛЕМЫ И СПОСОБЫ ИХ РЕШЕНИЯ:

4. **Данные на границе блоков памяти**(первые несколько байт переменной могут располагаться в одном блоке, а несколько – в следующем)
    a. **Выравнивание данных** – это изменение положения переменных в памяти таким образом, чтобы они были выровнены относительно некоторой величины(размера переменной или блока):
        i. Может производиться компилятором (автоматически или с ключами);
        ii. При динамическом выделении памяти можно пользоваться _mm_malloc или posix_memalign для выравнивания;
        iii. Можно также выделить область памяти большего размера с помощью стандартной функции malloc и сделать в начале области необходимый для выравнивания отступ;

iv. Нужно осторожнее работать с указателями, так как можно получить невыровненные данные.

5. **Разреженное размещение данных в памяти**(данные«разбросаны» по памяти на значительные расстояния друг от друга (больше размера кэш-строки))
    a. Разрешить компилятору производить выравнивание данных (обычно он это делает по умолчанию);
    b. Избегать особых ситуаций явного сдвига адреса элемента на значение, не кратное размеру элемента (как в листинге 8);
    c. Группировать данные в соответствии с их использованием;
    d. Размещать данные в структурах как можно более плотно.
6. **Данные, смещенные на величину, кратную размеру банка кэш-памяти**

    Для того, чтобы избежать эффекта **«буксования»** кэш-памяти, обычно используются два приема:
    1. Изменение порядка обхода элементов;
    2. Изменение расстояния между элементами(добавить фиктивные элементы в массив(не меньше размера кэш-строки)).

2. **Нарушение локальности во времени обращений в память**(блок памяти понадобился, но уже вытеснен из кэша другими блоками)
    1. Для решения этой проблемы программу перестраивают таким образом, чтобы если некоторая переменная была загружена из оперативной памяти в кэш-память, то над ней выполняются все возможные вычисления, пока она еще находится в кэш-памяти.
3. **Неупорядоченный обход данных в памяти**

    Способы достижения последовательного обхода памяти в программе:
    1. Перестановка циклов;

2. Изменение порядка следования размерностей массива(транспонировать матрицу).

Источник: методичка, стр. 99 - 115.

19. СРАВНИТЬ ЦЕНУ ПРОМАХА В КЭШ-ПАМЯТИ И В ВИРТУАЛЬНОЙ ПАМЯТИ.

ОТВЕТ:

	Кэш-память	**Виртуальная память**
Скорость доступа при попадании H:	Если брать среднее количество тактов по всем уровням кэша, то порядка **15** тактов.	Примерно **200** тактов.
Обращение за недостающими данными в более низкие уровни памяти L:	Обращается в оперативную память. Если допустить, что необходимые данные уже в оперативной памяти, то доступ к ней займет порядка **200** тактов.	Обращается в жесткий диск. Порядка 10^6 тактов.
Шанс промаха R:	Около 10%	0.001%

Введем величину V, отражающую цену промаха. Понятно, что она прямо-пропорциональна шансу промаха R, потому что чем чаще происходит промах, тем медленнее работает память. V прямо-пропорциональна $\frac{L}{H}$, потому что чем медленнее более низкий уровень памяти по сравнению с более высоким, тем больший вклад вносится в цену промаха. Почему не используются такие показатели памяти, как частота обращения и объем? Эти показатели “съедаются” шансом промаха R. Действительно, чем выше шанс промаха, тем выше частота обращений(просто проводится больше “экспериментов” и у быстрой памяти нет большого запаса времени на оптимизацию хит рейта) и тем меньше объем(данных

отобразить в быструю память надо много, а ее мало, вот и происходят частые промахи).

Таким образом, $V = R * \frac{L}{H}$

V кэш-памяти: V = 10 * $\frac{200}{15} \approx 133$

V виртуальной памяти: V = 0.001 * $\frac{1000000}{200} = 5$

Таким образом, **цена промаха кэш-памяти выше, чем цена промаха виртуальной памяти**, во многом благодаря низкому мисс рейту виртуальной памяти.

Источники: брошюры университета Айовы по кэшу и по виртуальной памяти, пирамида иерархии памяти сверху, а так же здравый смысл.

20. СПОСОБЫ ОПТИМИЗАЦИИ ПРОГРАММ, ИСПОЛЬЗУЕМЫЕ КОМПИЛЯТОРАМИ.

ОТВЕТ:

a. **Удаление мертвого кода** – преобразование, удаляющее фрагменты кода, которые не влияют на результат программы;
b. **Отображение переменных на регистры процессора**;
c. **Раскрутка циклов** включается ключами GCC -funroll-loops, -funroll-all-loops. В результате этого преобразования исходный цикл преобразуется в другой цикл, в котором тело цикла содержит несколько тел старого цикла (листинг 1). При этом счетчик цикла меняется соответственно;
d. **Встраивание функций.** При использовании этого преобразования вместо вызова функции в код встраивается тело функции. При этом ценой увеличившегося размера кода устраняются временные издержки на вызов функции и передачу аргументов;

e. **Переупорядочивание команд.** Команды переупорядочиваются с учетом информационных зависимостей таким образом, чтобы более равномерно и полно загружать вычислительные устройства процессора;

f. **Вынос инвариантных вычислений за циклы.** Если в цикле присутствуют вычисления, которые не зависят от итерации цикла, то они выносятся за цикл и тем самым многократно не повторяются;

g. **«Перепрыгивание» переходов.** Если в программе имеется цепочка последовательных переходов (условных или безусловных), она заменяется на единственный переход, который ведет сразу в окончательный пункт назначения, минуя промежуточные переходы;

h. **Устранение несущественных проверок указателей на NULL.**

Источник: методичка, стр. 73 - 76.

21. НАБОР КОМАНД ПРОЦЕССОРА, ТРЕБОВАНИЯ К НАБОРУ КОМАНД

Необходимо учитывать
•Общее число различных команд.
•Общую длину команды.
•Тип полей и их длина.
•Простота декодирования.
•Адресуемость и способы адресации.

Команда процессора - элементарная операция, выполняемая процессором.
Набор команд - множество команд, которые процессор способен выполнять.
Источник: презентация про набор команд, методичка.

22. ПРОЦЕССОР, ЕГО СОСТАВ И ФУНКЦИОНИРОВАНИЕ. ТЕХНИКА КОНВЕЙЕРИЗАЦИИ. ПЕРЕДАЧА ДАННЫХ НА КОНВЕЙЕРЕ. УВЕЛИЧЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ ЗА СЧЕТ КОНВЕЙЕРИЗАЦИИ.

Большая часть процессоров в наше время основаны на вм фон Неймана.
Таким образом, характерной чертой компьютеров фон Неймана является наличие **глобально адресуемой памяти** и **счетчика команд**, которые позволяют УУ многократно повторять один и тот же цикл действий:

извлечение очередной команды машинного кода,декодирование и выполнение команды
в автоматическом режиме. В результате глобально адресуемая память и счетчик команд **создают поток команд**, которые УУ декодирует, а АЛУ исполняют.
Про конвейер смотри 23 вопрос, https://pandia.ru/text/78/353/127.php

23. КОМАНДНЫЙ КОНВЕЙЕР. ПРИМЕР КОМАНДНОГО КОНВЕЙЕРА. СПОСОБЫ УВЕЛИЧЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ КОНВЕЙЕРА. ПРИЧИНЫ ПРИОСТАНОВКИ КОНВЕЙЕРА И ТЕХНИКА ИХ ПРЕОДОЛЕНИЯ.

ОТВЕТ:

Конвейеризация – это техника, в результате которой задача или команда разбивается на некоторое число подзадач, которые выполняются последовательно. Каждая подкоманда выполняется на своем логическом устройстве. Все логические устройства (ступени) соединяются последовательно таким образом, что выход i-ой ступени связан с входом (i+1)-ой ступени, все ступени работают одновременно. Множество ступеней называется конвейером. Выигрыш во времени достигается при выполнении нескольких задач за счет параллельной работы всех ступеней, вовлекая на каждом такте новую задачу или команду.

Таким образом, выполнение команды разделяется на несколько стадий, каждая из которых выполняется на соответствующей ступени конвейера. **Типичный набор стадий выполнения команды:** 1) Выборка команды

2) Декодирование команды

3) Выборка и загрузка операндов

4) Выполнение операции

5) Сохранение результатов в память

Причины низкой производительности или приостановки конвейера и техника их преодоления:

4. **Зависимости по данным**(когда команда уже может получить операнды, но над этими операндами работает выполняющаяся команда):
    1. **Data forwarding-** это техника ускоренной передачи результата одной операции на другую операцию через процессорные регистры без записи результата в ячейку памяти;
    2. **Разнесение зависимых по данным команд**, заполняя пространство между ними другими независимыми от них командами.
5. **Конфликты по ресурсам**(на некоторой стадии команда должна обратиться к некоторому ресурсу процессора, который занят другой командой):
    1. Для устранения зависимостей по ресурсам на уровне микроархитектуры обычно используется **дублирование ресурсов.** Например, в процессоре может быть несколько функциональных устройств, которые работают одновременно. Один регистровый файл может быть разбит на два (или даже продублирован), доступ к которым осуществляется независимо. На уровне кода конфликтующие по ресурсам команды обычно разносятся компилятором на некоторое расстояние, и между ними вставляются другие, независящие от них команды.
6. **Зависимости по управлению**, вызванные командами перехода(на конвейер попала команда, которая не должна быть выполнена из-за обработанного только что ветвления в программе - придется делать сброс конвейера, начиная работу с команды, которая была выбрана ветвлением):
    1. **Механизм раннего обнаружения и предсказания переходов:**

1. **Статические методы** используют информацию из кода программы, специально выработанную компилятором. При использовании этого способа процессор делает однозначный вывод о срабатывании команды перехода по ее виду. Статическое предсказание срабатывает **на стадии декодирования команды**;
2. **Динамический** (учитывает предысторию выполнения программы, она записывается в таблицу истории переходов (Branch History Table (BHT)). В строке таблицы для команды условного перехода содержится целевой адрес и бит, который указывает, был ли сделан переход, когда команда встретилась последний раз. Если прогноз не верен, то бит в таблице меняется. Динамическое предсказание срабатывает уже **на стадии выборки команды**, т.к. команды перехода в таблицах истории идентифицируются по своему адресу.

Источник: методичка, стр. 44 - 50.

24. СПОСОБЫ ПРЕДСКАЗАНИЯ ПЕРЕХОДОВ. ВИДЫ ДИНАМИЧЕСКИХ ПРЕДСКАЗАТЕЛЕЙ ПЕРЕХОДОВ.

См. Вопрос 30 про статические и динамические методы.
https://ru.wikipedia.org/wiki/Предсказатель_переходов
Динамические методы, широко используемые в современных процессорах, подразумевают анализ истории ветвлений.
Счётчик с насыщением или бимодальный счётчик

Анализируется таблица истории переходов. Таблица содержит:

младшие значимые биты адреса инструкции;
соответствующую им вероятность условного перехода:
«скорее всего, будет выполнен»;
«возможно, будет выполнен»;
«возможно, не будет выполнен»;
«скорее всего, не будет выполнен».

Таблица обновляется после каждого перехода. Алгоритм изменяет выбор ветвления, если результат условия отклонился два раза от предыдущих результатов. Использование младших битов адреса инструкции позволяет производить предсказания нескольких инструкций до их декодирования.

Адаптивный двухуровневый предсказатель
Для первого уровня выполняются история последних k ветвлений, второго уровня k указывает на таблицу шаблонов.

Локальное предсказание перехода
Каждый условный переход в области имеет собственную историю переходов. Шаблоны переходов могут быть общими или отдельными.

Глобальное предсказание перехода
Глобальное предсказание переходов не хранит истории отдельно для каждого перехода, а использует общую историю. Любые закономерности в переходах сказываются на этой истории, но историю могут загрязнять нерелевантные записи.

Гибридный предсказатель
Гибридный предсказатель может выбирать результаты наиболее успешных предсказателей на основе истории либо использовать мажоритарную функцию нечётного количества предсказателей.

Предсказатель для цикла
Предсказатель для цикла может использовать счетчик цикла для отсчета количества переходов в начало цикла. Этот предсказатель может использоваться в гибридном предсказателе.

Предсказание косвенных переходов
Косвенный переход может иметь больше двух ветвлений. Новейшие процессоры имеют возможность выбора более двух условий используя двухуровневый адаптивный предсказатель. Процессоры без поддержки предсказаний косвенных переходов могут использовать статическое предсказание или брать предыдущее значение.

25. КЛАССИФИКАЦИЯ ПЕРИФЕРИЙНЫХ УСТРОЙСТВ, НАЗНАЧЕНИЕ И ОСНОВНЫЕ ХАРАКТЕРИСТИКИ КАЖДОГО ВИДА ПЕРИФЕРИЙНЫХ УСТРОЙСТВ.
    - Устройства ввода типа мышка, клавиатура, графический планшет, сканер, геймпад, камера, микрофон и т.п.

- Устройства вывода типа монитор, проектор, принтер, наушники, динамики и т.п.
- Устройства хранения информации, которые и на ввод и на вывод одновременно, например, диск, флешка, минифлешка, магнитофон (внезапно)
- Устройства передачи информации - модемы, факсы, блютус.

26. ВИДЫ ШИН В ЭВМ, ИХ НАЗНАЧЕНИЕ И ОСНОВНЫЕ ХАРАКТЕРИСТИКИ.

ОТВЕТ:

Виды шин:

7. **Шина «процессор-память»:**

   Шина «процессор-память» обеспечивает непосредственную связь между центральным процессором (ЦП) вычислительной машины и основной памятью (ОП). В современных микропроцессорах такую шину часто называют шиной переднего плана и обозначают аббревиатурой FSB (Front-Side Bus). Интенсивный трафик между процессором и памятью требует, чтобы полоса пропускания шины, то есть количество информации, проходящей по шине в единицу времени, была наибольшей. Роль этой шины иногда выполняет системная шина (см. ниже), однако в плане эффективности значительно выгоднее, если обмен между ЦП и ОП ведется по отдельной шине. К рассматриваемому виду можно отнести также шину, связывающую процессор с кэш-памятью второго уровня, известную как шина заднего плана — BSB (Back-Side Bus). BSB позволяет вести обмен с большей скоростью, чем FSB, и полностью реализовать возможности более скоростной кэш-памяти.

8. **Шины ввода/вывода:**

   Шина ввода/вывода служит для соединения процессора (памяти) с устройствами ввода/вывода (УВВ). Учитывая разнообразие таких устройств, шины ввода/вывода унифицируются и стандартизируются. Связи с

большинством УВВ (но не с видеосистемами) не требуют от шины высокой пропускной способности. При проектировании шин ввода/вывода в учет берутся стоимость конструктивна и соединительных разъемов. Такие шины содержат меньше линий по сравнению с вариантом «процессор-память», но длина линий может быть весьма большой. Типичными примерами подобных шин могут служить шины PCI и SCSI.

9. **Системные шины:**

   Системная шина служит для физического и логического объединения всех устройств ВМ. Поскольку основные устройства машины, как правило, размещаются на общей монтажной плате, системную шину часто называют объединительной шиной (backplane bus), хотя эти термины нельзя считать строго эквивалентными.

   Системная шина в состоянии содержать несколько сотен линий. Совокупность линий шины можно подразделить на три функциональные группы:

   4. **шина данных**;
   5. **шина адреса**;
   6. **шина управления**;

Основные характеристики шин:

4. **Разрядность шины** определяется числом параллельных проводников, входящих в нее;
5. **Тактовая частота**;
6. **Пропускная способность** шины определяется количеством байт информации, передающейся в шине за секунду. Для определения пропускной способности шины необходимо умножить тактовую частоту шины на её разрядность.

Источник: методичка СибГУТИ.

27. ОРГАНИЗАЦИЯ ВЗАИМОДЕЙСТВИЯ НЕСКОЛЬКИХ УСТРОЙСТВ НА ШИНЕ. РАЗДЕЛЕНИЕ НА ВЕДУЩИЕ И ВЕДОМЫЕ УСТРОЙСТВА. МЕХАНИЗМ ПРЕРЫВАНИЙ. АРБИТРАЖ ШИН И СХЕМЫ АРБИТРАЖА.

ОТВЕТ:

Прерывания - 28 вопрос.

Шина - это коммуникационное аппаратное обеспечение, представляющее собой набор проводников, несущих двоичные сигналы.

Шинный интерфейс – иерархия шин вместе с их протоколами работы, которые обеспечивают связь процессора с кэш памятью, оперативной памятью и периферийными устройствами.

Функции шин:

4. Передача данных и команд;
5. Организация взаимодействия устройств на шине;
6. Обнаружение и обработка ошибок в программном обеспечении или аппаратуре.

Когда два устройства обмениваются информацией по шине, одно из них должно инициировать обмен и управлять им. Такого рода устройства называют **ведущими** (bus master). В компьютерной терминологии «ведущий» — это любое устройство, способное взять на себя владение шиной и управлять пересылкой данных. Ведущий не обязательно использует данные сам. Он, например, может захватить управление шиной в интересах другого устройства. Устройства, не обладающие возможностями инициирования транзакции, носят название **ведомых** (bus slave). В принципе к шине может быть подключено несколько потенциальных ведущих, но в любой момент времени активным может быть только один из них: если несколько устройств

передают информацию одновременно, их сигналы перекрываются и искажаются. Для предотвращения одновременной активности нескольких ведущих в любой шине предусматривается процедура допуска к управлению шиной только одного из претендентов **(арбитраж)**. В то же время некоторые шины допускают широковещательный режим записи, когда информация одного ведущего передается сразу нескольким ведомым (здесь арбитраж не требуется). Сигнал, направленный одним устройством, доступен всем остальным устройствам, подключенным к шине. Организация взаимодействия устройств на шине осуществляется механизмом арбитража.

Типы арбитража:

3. Централизованный;
4. Децентрализованный.

	Централизованный	**Децентрализованный**
Кем осуществляется?	Специальное устройство - **центральный арбитр**, - ответственное за предоставление доступа к шине только одному из запросивших ведущих.	Каждый ведущий содержит блок управления доступом к шине, и при совместном использовании шины такие блоки взаимодействуют друг с другом, разделяя между собой ответственность за доступ к шине.
Способ подключения ведущих устройств к арбитру?	**Параллельный**: центральный арбитр связан с каждым потенциальным ведущим индивидуальными двухпроводными трактами. Запросы к	Нет, так как нет центрального арбитра.

	центральному арбитру могут поступать независимо и параллельно. **Последовательный**: это такой арбитраж, при котором все ведущие устройства передают сигнал разрешения захвата шины по цепочке от более приоритетных устройств к менее приоритетным до тех пор, пока он не достигнет ведущего, посылавшего сигнал запроса на захват шины. Изначально сигнал формируется центральным арбитром.	
Отказоустойчивость?	Центральный арбитр - единственная точка отказа системы.	Система устойчива к отказам.

Источник: лекции, методичка СибГУТИ.

28. МЕХАНИЗМЫ ОБРАБОТКИ ПРЕРЫВАНИЙ В ПРОЦЕССОРЕ.

ОТВЕТ:

Прерывание (англ. interrupt) — сигнал от программного или аппаратного обеспечения, сообщающий процессору о наступлении какого-либо события, требующего немедленного внимания. Прерывание извещает процессор о наступлении высокоприоритетного события, требующего прерывания текущего кода, выполняемого процессором. Процессор отвечает приостановкой своей текущей активности, сохраняя свое состояние и выполняя функцию, называемую

обработчиком прерывания (или программой обработки прерывания), которая реагирует на событие и обслуживает его, после чего возвращает управление в прерванный код.

В зависимости от источника возникновения сигнала прерывания делятся на:

10. **Асинхронные, или внешние** (аппаратные) — события, которые исходят от внешних аппаратных устройств (например, периферийных устройств) и могут произойти в любой произвольный момент: сигнал от таймера, сетевой карты или дискового накопителя, нажатие клавиш клавиатуры, движение мыши. Факт возникновения в системе такого прерывания трактуется как запрос на прерывание (англ. Interrupt request, IRQ) - устройства сообщают, что они требуют внимания со стороны ОС;
11. **Синхронные, или внутренние** — события в самом процессоре как результат нарушения каких-то условий при исполнении машинного кода: деление на ноль или переполнение стека, обращение к недопустимым адресам памяти или недопустимый код операции;
12. **Программные** (частный случай внутреннего прерывания) — инициируются исполнением специальной инструкции в коде программы. Программные прерывания, как правило, используются для обращения к функциям встроенного программного обеспечения (firmware), драйверов и операционной системы.

До окончания обработки прерывания обычно устанавливается запрет на обработку этого типа прерывания, чтобы процессор не входил в цикл обработки одного прерывания. Приоритизация означает, что все источники прерываний делятся на классы и каждому классу назначается свой уровень приоритета запроса на прерывание.

3. **Относительное обслуживание прерываний** означает, что если во время обработки прерывания поступает более приоритетное прерывание, то это прерывание будет обработано только после завершения текущей процедуры обработки прерывания;
4. **Абсолютное обслуживание прерываний** означает, что если во время обработки прерывания поступает более приоритетное прерывание, то текущая процедура обработки прерывания вытесняется, и процессор начинает выполнять обработку вновь поступившего более приоритетного прерывания. После завершения этой процедуры процессор возвращается к выполнению вытесненной процедуры обработки прерывания.

Источник: Википедия.

29. ПРОЦЕССОР, ЕГО СОСТАВ И ФУНКЦИОНИРОВАНИЕ. ПРИЧИНЫ ОСТАНОВКИ КОНВЕЙЕРА. ТЕХНИКА КОНВЕЙЕРИЗАЦИИ. ОЦЕНКИ СЛОЖНОСТИ РЕАЛИЗАЦИИ КОНВЕЙЕРА.
30. СТАТИЧЕСКИЙ И ДИНАМИЧЕСКИЙ СПОСОБЫ ПРЕДСКАЗАНИЯ ПЕРЕХОДОВ.

Динамическое предсказание перехода - предсказание о наиболее вероятном исходе команды условного перехода, которое принимается, исходя из собираемой в процессе выполнения программы информации о предшествующих переходах (методичка стр. 155)

Статическое предсказание перехода - предсказание наиболее вероятного результата команды условного перехода, закодированное в коде команды. Статическое предсказание основывается на некотором априорном знании компилятора о ходе выполнения программы. (методичка стр. 162)

Тут говорится, что статическое предсказание в наше время используется только тогда, когда динамическое использовать невозможно. Видимо, это те случаи, когда

нельзя собрать статистику исполнения программы - она мала или очень неоднородная и динамическое предсказание оказывается мало- или вообще не-эффективным. Статические предсказания это примитивные штуки, например в современных процах реализовано такое предсказание - любой переход назад (на младшие адреса) будет выполнен и если такой переход встречается, то в конвейер загружаются инструкции, расположенные по адресу перехода. А любой переход вперед (на старшие адреса) предположительно, выполнен не будет а загружаются инструкции идущие после инструкции перехода. Такой метод используется в качестве “подстраховки” и очевидно, что динамический метод, если его можно применить, будет более эффективен. Но если например в программе всего один переход, то динамический метод не сработает вообще, а статический с вероятностью 50% предскажет то, что нужно.

31. ДИНАМИЧЕСКОЕ ПЕРЕИМЕНОВАНИЕ РЕГИСТРОВ.

ОТВЕТ:

Переименование регистров (англ. register renaming) — метод ослабления взаимозависимостей команд, применяемый в процессорах с внеочередным исполнением команд. Один из методов, применяемых в вычислительных конвейерах для реализации параллелизма на уровне команд.

В том случае, если в соответствии с двумя или более командами необходимо осуществить запись данных в один регистр, их корректное внеочередное исполнение становится невозможным (более поздняя команда не может быть обработана до завершения более ранней) даже в том случае, если при этом нет зависимости по данным. Такие взаимозависимости часто называют ложными (в случае истинной зависимости существует зависимость и по данным).

Так как количество архитектурных регистров обычно ограничено (например,

стандартно архитектура x86 предусматривает только восемь регистров общего назначения), вероятность возникновения ложных взаимозависимостей достаточно велика, что может привести к снижению производительности процессора. Переименование регистров представляет собой преобразование программных ссылок на архитектурные регистры в ссылки на физические регистры и позволяет ослабить влияние ложных взаимозависимостей за счёт использования большого количества физических регистров вместо ограниченного количества архитектурных (так, например, x86-совместимые процессоры архитектуры Intel P6 содержат 40 физических регистров[1]). При этом процессор отслеживает, состояние каких физических регистров соответствует состоянию архитектурных, а выдача результатов осуществляется в порядке, который предусмотрен программой.

Источник: Википедия.

32. ОРГАНИЗАЦИЯ ПАРАЛЛЕЛЬНОГО ВЫПОЛНЕНИЯ КОМАНД В СУПЕРСКАЛЯРНЫХ ПРОЦЕССОРАХ.

ОТВЕТ:

Суперскалярный процессор получает от компилятора программу в виде последовательности команд. Специальное устройство в процессоре, называемое диспетчером, динамически выявляет в этой последовательности независимые команды, которые затем распределяются по нескольким функциональным устройствам для параллельного выполнения.

	Суперскалярный
Чем выявляется ILP?	Диспетчером в процессоре
Как выявляется ILP?	Множество очередных последовательно идущих и ожидающих выполнения команд программы образуют так называемое «**окно просмотра**». В

	рамках этого окна диспетчер ищет готовые к выполнению команды, по несколько за такт, и отправляет их на выполнение к соответствующим функциональным устройствам. Команда считается готовой к выполнению, если все ее зависимости разрешены.
Как команды исполняются?	В процессоре с **упорядоченным выполнением команд** диспетчер рассматривает команды только в том порядке, в котором они идут во входной последовательности. В процессоре с **выполнением команд вне порядка** (ООО) диспетчер рассматривает команды в пределах всего окна просмотра. Для повышения производительности суперскалярный процессор вынужден разрешать ложные зависимости с помощью механизма **переименования регистров**.

Источник: методичка, стр. 56 - 58.

33. ЗА СЧЕТ ЧЕГО В СУПЕРСКАЛЯРНОМ ПРОЦЕССОРЕ ПОДДЕРЖИВАЕТСЯ КОРРЕКТНОЕ ВЫПОЛНЕНИЕ ПОСЛЕДОВАТЕЛЬНОЙ ПРОГРАММЫ.

ОТВЕТ: Корректное выполнение последовательной программы в суперскалярном процессоре поддерживается за счет того, что диспетчер процессора выявляет для распараллеливания только независимые команды, т.е порядок исполнения которых никак не влияет на конечный результат вычислений.

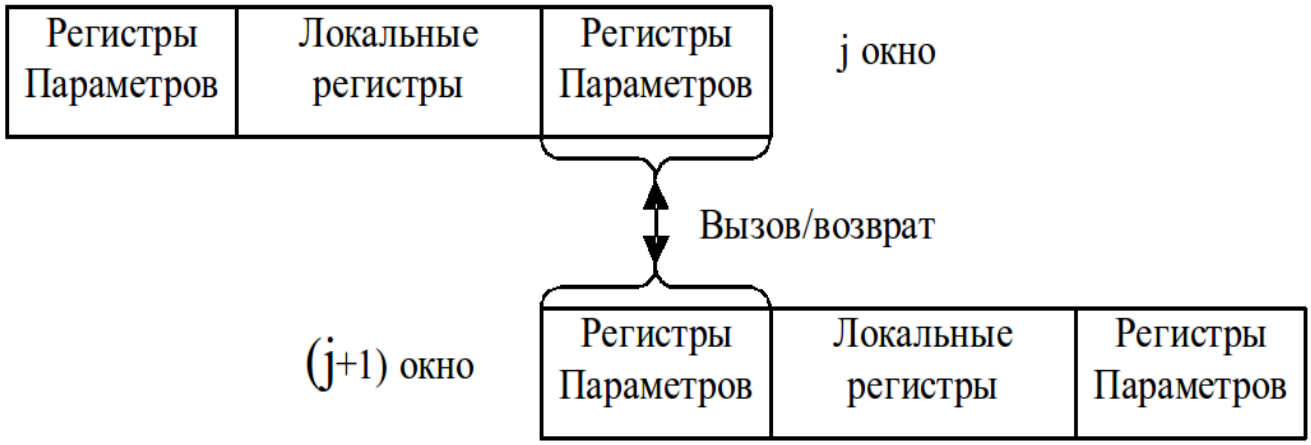
34. ФУНКЦИИ РЕГИСТРОВОГО ОКНА В RISC-ПРОЦЕССОРАХ.

ОТВЕТ:

Аппаратная оптимизация использования регистров сводится к сокращению затрат времени на работу с процедурой. Чтобы упростить и ускорить передачу параметров

от вызывающей процедуры к вызываемой и обратно, было предложено использовать **регистровые окна**. Для этого все множество регистров(регистровый файл) разбивается на подмножества, называемые регистровыми окнами, и каждому окну соответствует одна процедура. В каждый момент времени только одно окно доступно и адресуемо. Все окна имеют один и тот же размер и состоят трех полей (областей фиксированного размера):

4. **Левое поле** окна включает регистры, которые содержат параметры процедуры, которая вызвала текущую процедуру, и результаты, которые необходимо передать обратно.
5. **Среднее поле** представляет собой локальные регистры, которые используются для хранения локальных переменных, как определено компилятором, и констант процедуры.
6. **Правое поле** (временные регистры) используются для обмена параметрами и результатами с процедурой, вызванной текущей процедурой.



Источник: лекции. Тут чуть-чуть понятно.

35. Пример микропроцессора. Структура, организация конвейера, подсистемы памяти. К какому классу принадлежит.
36. Основные характеристики CISC-архитектуры. Формирование концепции RISC-архитектуры.

37. ПОНЯТИЕ РЕГИСТРОВОГО ОКНА.

ОТВЕТ: смотреть вопрос 34.

38. Оптимизирующий компилятор в RISC-процессорах.
39. СРАВНЕНИЕ CISC И RISC-АРХИТЕКТУР.

CISC:

- нефиксированное значение длины команды
- арифметические действия кодируются в одной команде;
- небольшое число регистров, каждый из которых выполняет строго определённую функцию.

RISC:

-Фиксированная длина машинных инструкций (например, 32 бита) и простой формат команды. *В киске команды разной длины и это хуже для конвейера, так как команды выполняются за различное число тактов и конвейер будет стоять и ждать, пока выполнится какая-нибудь сложная команда, в риске такой лажи нет.*

-Специализированные команды для операций с памятью — чтения или записи.
-Операции вида Read-Modify-Write («прочитать-изменить-записать») отсутствуют. Любые операции «изменить» выполняются только над содержимым регистров (т. н. архитектура load-and-store). *Простыми словами, тут сначала загружаешь в регистр, преобразовываешь и затем выгружаешь в память, в киске всё это можно сделать одной командой например.*

-Большое количество регистров общего назначения (32 и более).

-Отсутствие поддержки операций вида «изменить» над укороченными типами данных — байт, 16-разрядное слово. *Например, если регистр 64 бита, то в риске нельзя изменить первые 16 бит, или первые 32 бита, только через танцы с бубном. В киске это можно, там в зависимости от первой буквы в имени регистра (типа RAX, EAX) мы меняем те или иные биты.*

-Отсутствие микропрограмм внутри самого процессора. То, что в CISC-процессоре исполняется микропрограммами, в RISC-процессоре исполняется как обыкновенный (хотя и помещенный в специальное хранилище) машинный код, не отличающийся принципиально от кода ядра ОС и приложений.

40. Структура суперскалярного процессора. Причины, ограничивающие производительность суперскаляров, и средства их преодоления. Примеры микропроцессоров.
41. СТРУКТУРА VLIW ПРОЦЕССОРОВ. ПРИЧИНЫ, ОГРАНИЧИВАЮЩИЕ ПРОИЗВОДИТЕЛЬНОСТЬ ПРОЦЕССОРОВ VLIW, И СРЕДСТВА ИХ ПРЕОДОЛЕНИЯ. ПРИМЕРЫ МИКРОПРОЦЕССОРОВ.

ОТВЕТ:

На вход процессора подается последовательность больших команд, состоящих из нескольких простых операций, которые могут исполняться параллельно.

	VLIW
Чем выявляется ILP?	Компилятором
Как выявляется ILP?	Компилятор для VLIW-процессора выполняет статический анализ программы, выявляет независимые операции и формирует последовательность связок для выполнения VLIW-процессором. Число команд в связке определяется архитектурой процессора и равно ширине конвейера. Для каждой простой команды в коде указаны все необходимые аппаратные ресурсы для ее выполнения: аппаратные регистры, функциональные устройства. Если компилятор не нашел достаточное количество независимых команд для

	формирования связки, он дополняет ее командами NOP («нет операции»). По сути, компилятор выполняет почти полное детальное планирование выполнения потока команд на VLIW-процессоре.
Как команды исполняются?	Диспетчер VLIW-процессора только выполняет закодированные в программе указания, т.к все уже проанализировано и распланировано компилятором.

VLIW-процессор:

Плюсы:

1. Простая логика работы процессора по сравнению с суперскалярными процессорами;
2. На кристалле много места под ресурсы;
3. Ширина конвейера типичных VLIW-процессоров (6-8 команд) больше ширины конвейера типичных суперскалярных процессоров;
4. Производительнее суперскалярных.

Минусы:

1. Часть параллелизма не может быть выявлена вообще, поскольку у компилятора нет информации о зависимостях, которые формируются в процессе вычисления;
2. Производительность VLIW-процессоров сильно зависит от качества кода, а VLIW-код жестко «привязан» к конкретной микроархитектуре процессора.

Пример процессора: Itanium.

Источник: методичка стр. 59 - 61.

42. Сравнить способы реализации условного перехода в суперскалярах и VLIW-процессорах.
43. СРЕДСТВА ПОВЫШЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ В ПРОЦЕССОРЕ ITANIUM.

ОТВЕТ:

Способы увеличения ILP:

a. Явная спекуляция по данным и управлению (уменьшает задержки по памяти);
b. Предикатное исполнение команд (устраняет ветвления);
c. Аппаратная поддержка программной конвейеризации циклов (вращение регистров, специальные счетчики циклов, предикатные регистры);
d. Предсказание ветвлений;
e. Специальные векторные инструкции.

Специальные способы увеличения производительности:

a. Специальная поддержка модульности программ (регистровый стек, вращающиеся регистры);
b. Высокопроизводительная вещественная арифметика.

Источник: сайта нашего любимого вуза.

44. Общие и разные черты конвейеров процессоров Power4 и Itanium?
45. СПОСОБЫ ПОДДЕРЖАНИЯ КОГЕРЕНТНОСТИ ДАННЫХ В КЭШ-ПАМЯТИ.

ПРОБЛЕМА:

Когерентность кэша - свойство кэшей, означающее целостность данных, хранящихся в локальных кэшах для разделяемого ресурса. Проблема когерентности кэш-памяти характерна для многоядерных(многопроцессорных) систем. Например, если одно ядро(процессор) изменило данные и записало их обратно в кэш, но не в оперативку, а эти же данные потребовались второму ядру(процессору), то оно, запросив их у оперативки, вообще говоря рискует получить неактуальные данные.

Проблема, о которой идет речь, возникает из-за того, что значение элемента данных в памяти, хранящееся в двух разных ядрах(процессорах), доступно этим ядрам(процессорам) только через их индивидуальные кэши.

Когерентность определяет поведение чтений и записей в одно и то же место памяти. Кэш называется когерентным, если выполняются следующее условие:

Информирование о записи. Изменение данных в любом из кэшей должно быть распространено на другие копии этих данных (этой линии кэша) в соседних кэшах.

РЕШЕНИЕ:

4. **Когерентность с использованием справочника.** Информация о состоянии блока физической памяти содержится только в одном месте, называемом справочником (физически справочник может быть распределен по узлам системы).

   Протокол на основе “полного справочника” является наиболее простым из протоколов на основе директорий. Здесь присутствует единый централизованный справочник, хранящийся в основной памяти, который поддерживает информацию обо всех кэшах. Справочник содержит множество записей, описывающих каждую кэшируемую ячейку ОП. Обращение к справочнику производится каждый раз, когда один из процессоров изменяет копию такой ячейки в своей локальной памяти. В этом случае информация из справочника нужна для того, чтобы аннулировать или обновить копии измененной ячейки (или всей строки, содержащей эту ячейку) в других локальных кэшах, где присутствует эта же строка. Для каждого блока ОП в справочнике выделяется одна запись, хранящая указатели на копии данной

строки. Кроме того, в каждой записи выделен один бит модификации (D), показывающий, является ли копия “грязной”- (D = 1 - dirty) или “чистой”-(D = 0 - clean), то есть изменялось ли содержимое строки в кэш-памяти после того, как она была туда загружена. Этот бит указывает, имеет ли право процессор производить запись в данную строку.

5. **Когерентность с использованием отслеживания.** Каждый кэш, который содержит копию данных некоторого блока физической памяти, имеет также соответствующую копию служебной информации о его состоянии. Централизованная система записей отсутствует. Обычно кэши расположены на общей (разделяемой) шине и контроллеры всех кэшей наблюдают за шиной (просматривают ее) для определения того, не содержат ли они копию соответствующего блока.

Протокол **MESI:**

Протокол MESI реализован в современных многоядерных процессорах Intel и архитектуре PowerPC. Каждая строка кэша, согласно протоколу MESI, может находиться в одном из **четырех состояний**:

Modified - строка в кэше была изменена (отлична от основной памяти) и новое значение доступно только в данном кэше;

Exclusive - значение строки совпадает со значением в основной памяти, но его нет ни в одном другом кэше;

Shared - значение строки совпадает со значением в основной памяти и может присутствовать в других кэшах;

Invalid - строка кэша содержит недостоверную информацию.

6. **Перехват.** Когда из какого-либо одного кэша данные переписываются в оперативную память, контроллеры остальных получают сигнал об этом изменении ("перехватывают" информацию об изменении данных) и, если необходимо, изменяют соответствующие данные в своих кэшах.

Источники: Википедия, методичка Донецкого универа.

46. СРАВНИТЬ РЕАЛИЗАЦИЮ КОНВЕЙЕРА У VLIW ПРОЦЕССОРОВ И СУПЕРСКАЛЯРНЫХ ПРОЦЕССОРОВ.

ОТВЕТ: в общем, в суперскалярах конвейер сложный, потому что ему нужно подбирать команды, которые не пересекаются по различным параметрам, типа данных или ресурсов, выкидывать команды, которые оказались неверно предсказанными и т.п. VLIW ничем таким не занимается, за него всё это делает компилятор, он формирует последовательность из связок независимых команд, по длине равных ширине конвейера. Если компилятор не нашёл достаточное количество независимых команд, то он просто забивает связку NOP-ами (“нет операции”), то есть компилятор выполняет всё планирование выполнения потока команд, что в суперскалярах делает сам процессор. Это упрощает логику работы процессора, в результате на кристалле больше места для ресурсов и из-за этого ширина конвейера типичных VLIW - процессоров (6 - 8 команд) больше ширины конвейера типичных суперскалярных процессоров (3 - 5 команд). Совсем напрямую: в суперскаляре конвейер умный и медленный, в VLIW конвейер тупой и быстрый.

(Источник: методичка стр. 60, тут какие-то ошметки информации есть)

47. ПРОГРАММНЫЙ СПОСОБ КОНВЕЙЕРИЗАЦИИ ЦИКЛОВ.

ОТВЕТ:

Программная конвейеризация циклов - это техника, используемая компиляторами для оптимизации циклов по аналогии с вычислительным конвейером в микропроцессорах. Является формой внеочередного исполнения с той разницей, что переупорядочивание выполняется не процессором, а компилятором (либо, в случае ручной оптимизации, программистом). Некоторые компьютерные архитектуры, например Intel IA-64, имеют явную аппаратную поддержку для упрощения программной конвейеризации циклов. При конвейеризации цикла в каждый момент времени на исполнении находится код, относящийся к нескольким итерациям цикла, но к различным частям цикла.

Следующие аппаратные решения упрощают описанную оптимизацию:

13. **«Вращающиеся регистры»** - часть регистрового файла отводится на область вращающихся регистров. Инструкции, использующие некоторый архитектурный регистр из этой области, будут обращаться к различным физическим регистрам по мере исполнения итераций (и продвижения вращающейся области). Через какое-то количество итераций вновь произойдет обращение к исходному физическому регистру. Это позволяет работать с различными итерациями цикла одновременно, и при этом не требуется явных пересылок между регистрами;
14. **Предикаты и предикатное исполнение инструкций**, при котором предикатом работает некоторые специальные предикаты цикла. Эти предикаты позволяют включать и отключать некоторые инструкции цикла в процессе прохождения итераций, тем самым реализуя пролог и эпилог цикла в основном его коде, а также упрощают раскрытие условных операций в теле цикла;

15. **Аппаратная поддержка циклов**, при которой программа дает процессору информацию о размере цикла и о параметрах индексной переменной. Это позволяет сократить накладные расходы на организацию цикла. Также позволяет настроить скорость вращения и размер группы вращающихся регистров.

Источник: Википедия, методичка по программной конвейеризации циклов на платформе ARM от ИСП РАН.

48. Общие черты у RISC-процессоров и VLIW-процессоров.

https://pandia.ru/text/78/353/127.php

P.s. Почему не дописаны вопросы? А потому что, когда мы пришли сдавать Марковой, то обнаружили, что этой мамзель абсолютно всё равно на какие-то билеты, она доставала вопросы из глубин своего воспаленного сознания, при этом если задумаешься хотя бы на несколько секунд, то “не выше тройки, по глазам вижу, что ничего не знаете”. Стало очевидно, что ей можно сдать только путём хитрости и обмана, но никак не с помощью знаний, так и сдали.