1. Архитектура, микроархитектура: отличия, примеры

Архитектура (Instruction Set Architecture, ISA) – набор инструкций, регистров, режимов адресации, организации виртуальной памяти, обработки исключений и прерываний. То есть все то, что представляет процессор со стороны программиста. («Старый» взгляд на компьютерную архитектуру)

«Реальная» архитектура компьютера: включает в себя ISA, микроархитектуру, аппаратное обеспечение.

ISA – это интерфейс между CPU и программами, которые на этом CPU исполняются. ISA определяет количество регистров, которые видны программисту, доступные инструкции, правила выравнивания и так далее. В целом ISA - абстракция для программиста.

Микроархитектура – способ, которым данная архитектура набора команд реализована в процессоре.

Микроархитектура определяет то, как CPU устроен внутри. Из каких функциональных узлов он состоит, какая иерархия кэшей, как декодируются инструкции, сколько стадий в конвейере и так далее.

1. Микроархитектура, не влияет на семантику и результат исполнения программ, но влияет на скорость их исполнения.
2. Микроархитектура влияет на энергопотребление процессора. Например, Intel запустила линейку процессоров Atom, которая была нацелена на мобильные устройства. Эти CPU реализовали ту же архитектуру x86, но их внутреннее устройство позволяло экономить потребление энергии, что делало возможным их использование в переносимых устройствах.

Микроархитектура программисту не видна, программист работает на уровне ISA.

Если два разных процессора реализуют одну и ту же ISA, то они могут исполнять одни и те же программы. Например, программы можно запускать как на Intel, так и на AMD.

- Архитектура / Instruction Set Architecture (ISA)
  - programmer visible state (память & регистры)
  - доступные операции (инструкции и как они работают)
  - семантика исполнения (прерывания)
  - ввод / вывод
  - типы данных и их размеры
- Микроархитектура / Microarchitecture
  - исследования, как улучшить ISA по той или иной метрике
  - длина конвейера, размер кэша, набор функциональных блок и их взаимодействие

2. Производительность процессора: понятие “производительность”, способы измерения, единицы измерения

Производительность – это характеристика, которая отражает способность процессора выполнять вычисления за определенное время. Она

определяется скоростью и эффективностью выполнения инструкций, обработки данных и взаимодействия с другими компонентами системы.

Типичные показатели производительности – время отклика и пропускная способность (объем данных, обрабатываемых за единицу времени.).

Способы измерения:

1) Теоретические методы (оценка на основе технических характеристик):

- тактовая частота;
- количество операций на такт (CPI – Cycles Per Instruction);
- количество ядер и их взаимодействие.

2) Бенчмарки (задачи, служащие эталонным тестом производительности): синтетические (оценивают процессор в искусственных задачах):

- Dhrystone – производительность целочисленных вычислений, измеряемая в DMIPS;
- NBench – для измерения производительности целочисленных вычислений, операций памяти и вычислений с плавающей точкой.
- SPECo6fp – оценивает производительность вычислений с плавающей запятой, которые часто используются в научных, инженерных и других вычислительно сложных задачах, таких как моделирование, обработка изображений и симуляции.

Единицы измерения:

1) Количество операций в секунду:

- FLOPS (Floating Point Operations Per Second) – операции с плавающей запятой.
- MIPS (Million Instructions Per Second) – миллионы инструкций в секунду.

2) Время выполнения задач. Время выполнения реальных задач, таких как рендеринг, компиляция или математические расчеты.

3) Балл производительности. Результаты тестов в бенчмарках.

Например:

- Dhrystone: Процессор с результатом 5000 DMIPS способен выполнять 5 миллиардов целочисленных операций в секунду.
- NBench: В тесте Integer Index процессор набрал 150 баллов, что указывает на его высокую скорость обработки целочисленных операций.
- SPECo6fp: Для задач с плавающей запятой процессор может достичь 40 баллов в Base режиме, что делает его подходящим для научных вычислений.

3. Классификация архитектур (CISC, RISC, VLIW, EPIC)

Способы классификации архитектур:

- Control Flow / Data Flow архитектуры;
- по способу хранения операндов: стековая, аккумуляторная (IBM 7090, DEC PDP-8, Агат-8), регистровая;
- по доступу к операндам
- по типу архитектуры: CISC, RISC, VLIW, MIPS, PowerPC, SPARK, x86, x86-64, ARM

CISC архитектура (Complex Instruction Set Computer)
- Малое количество регистров
- Большое количество режимов адресации
- Сложные инструкции
- Переменная длина инструкции

RISC архитектура (Reduced instruction set computer)
- архитектура загрузки сохранения (load-store)
- большое количество универсальных регистров (register-register)
- простые инструкции
- ограниченное число режимов адресации
- инструкции фиксированной длины

Причины перехода от CISC к RISC: конвейеризация процессора, увеличение объема памяти, совершенствование компилятора

VLIW (Very Long Instruction Word)

Архитектура, которая специально разработана для повышения количества одновременно исполняемых инструкций. При одинаковом количестве одновременно исполняемых инструкций VLIW архитектура позволяет создать значительно более простой процессор, чем RISC архитектура.

Ключевая идея:

1. Множество НЕЗАВИСИМЫХ инструкций группируются в ОДИН пакет (VLIW инструкцию)
2. На одном такте выбирается ОДИН пакет
3. Каждая инструкция в пакете соответствует строго определенному функциональному устройству (ФУ)
4. Между всеми ФУ делится общий регистровый файл
5. Типовая длина пакета – 64-1024 бит (в зависимости от числа ФУ)
6. Проверка зависимостей и планирование исполнения возлагается на компилятор

Плюсы:
- аппаратная сложности уменьшена
- отсутствуют блок проверки зависимостей

- существенно упрощены блоки планирования исполнения и декодирования инструкций
- как следствие, сниженное энергопотребление и повышение тактовых частот
-

Минусы:
- высокие требования к компилятору
- увеличенный размер программного кода
- более высокие требования к пропускной способности памяти и регистрового файла
- незапланированные события (например, кэш-промах) приводят к останову конвейера
- отсутствие бинарной совместимости

EPIC (explicit parallel instruction set) – призвана сохранить достоинства VLIW архитектуры и устранить ее недостатки.

Общие черты VLIW и EPIC:
- возможность параллельного исполнения инструкций задается на уровне архитектуры, а не определяется динамически в процессе исполнения

Преимущества EPIC:
- решена проблема бинарной совместимости между различными процессорами

4. Классификация архитектур (x86, x86-64, Power, ARM, IA64, RISC-V, MIPS, Alpha, …)

Здесь важнее всего понимать, что все эти архитектуры отличаются только структурой инструкций, внутренней схемой соединения элементов, какими то другими конструктивными особенностями и системой команд. Умирали они потому, что у компанией заканчивались деньги/не успевали за конкурентами.

MIPS (умерло), SPARC (умирает)
Power (все сложно), 80x86 (все сложно)
ARM, RISC-V

MIPS

Область применения – встраиваемые решения 1981-2021 (после переход на RISC-V).

Характеристики архитектуры:

1. Малое количество простых инструкций
2. Всего 3 формата инструкций (3 способа расположения данных внутри инструкции)
3. Широкое распространение в академических кругах

Применялось в сетевых устройствах Cisco, Mikrotik, в старых консолях PlayStation, встраиваемые системы, цифровые приставки, кабельные модемы

Power PC

- архитектура POWER
  - Performance Optimization With Enhanced RISC
- область применения
  - высокопроизводительные серверные, настольные, встраиваемые решения
- 1991 г.: альянс AIM: Apple, IBM, Motorola
- 2013 г.: OpenPOWER Foundation

Юзали древние макбуки, вроде как иногда применяют военной промышленности, инфы очень мало.

80x86/x64

Область применения: высокопроизводительные настольные решения, сервера среднего и начального уровня, суперкомпьютеры

Основные производители процессоров: Intel, AMD, много мелких компаний из Китая

В 1978 выпущен 16 битный процессор 8086

Используют CISC, отличаются в объёме доступной ОЗУ, x64 обратно совместима с x86.

В x64 регистры 64 битные.

Различные режимы адресации (реальный, защищённый, смешанный).

Расширения SSE1..5, AVX, AVX2, AVX512 и другие подобные.

ARM (Advanced RISC Machine)

Низкое энергпотребление, поэтому юзаются во встраиваемых решениях. Единое адресное пространство для IO и ОЗУ

Надо покупать лицензию, чтобы использовать.

- Область применения:
  - встраиваемые решения (смартфоны, планшеты)
- Разработчик:
  - ARM Limited
- Разработка: с 1985 г. (ARM1)
- Семейства:
  - ARM7, ARM9, ARM11, Cortex, Neoverse

Alpha (умер юзал RISC)

- Область применения:
  - высокопроизводительные сервера и суперкомпьютеры
- Основные производители процессоров:
  - DEC -> Compaq (2001) -> HP (до 2007 г.)
- Все процессоры 64-х битные
- С 1992 по 2001 год процессоры Alpha являлись самыми быстрыми в мире
- В 1992 выпущен первый процессор Alpha 21064

PA-RISC (Precision Architecture, умер юзал RISC)

- область применения:
  - высокопроизводительные сервера и суперкомпьютеры
  - наиболее сложной и развитой системой команд среди всех RISC архитектур
- основные производители процессоров:
  - HP
- в 1989 выпущен первый 32-х битный процессор PA-7000
- в 1996 выпущен 64-х битный процессор PA-8000
- системы сняты с продаж в 2003 году, после перехода HP на процессоры Intel Itanium 2

IA-64 (умер, юзал RISC)

- область применения:
  - высокопроизводительные сервера и суперкомпьютеры
- разработка:
  - Intel + HP (с 1994 г.)
- архитектура относится к новому поколению EPIC
- ядра:
  - Intel Itanium (2001 г.)
  - Intel Itanium 2 (2002-2009 г.)
  - Itanium 9700 (2017 г.)

RISC-V:

RISC-V – это открытая архитектура, в отличие от большинства других RISC-архитектур, таких как ARM и MIPS, которые имеют лицензируемые патенты. Это позволяет производителям создавать свои собственные процессоры, используя RISC-V без платы за лицензии.

RISC-V отличается модульной структурой, что означает, что его можно настроить под конкретные потребности.

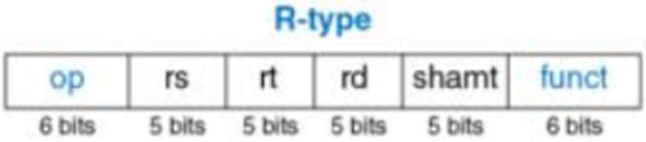
RISC-V активно развивается и поддерживается сообществом, в том числе такими крупными организациями, как Google, Western Digital и другие. Это означает, что архитектура постоянно улучшает свою совместимость с новыми технологиями.

5. Кодирование инструкций на примере MIPS и x86

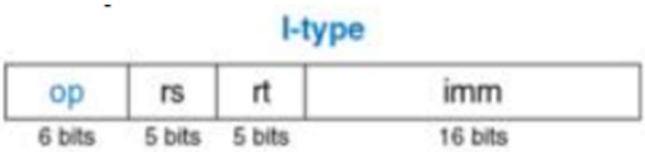
Инструкции бывают:

- Постоянной длины
  — простота декодирования
  — RISC: MIPS (4 байта), PowerPC, SPARC, ARM...
- Переменной длины
  — занимают меньше памяти
  — CISC: IBM 360, x86(от 1 до 17 байт), Motorola 68k, VAX...
- Комбинированный
  — MIPS16, THUMB (2 формата: 2 и 4 байта)

Особенность инструкций MIPS это фиксированная длина (4 байта). В архитектуре MIPS в качестве компромисса используются три формата инструкций: типа R, типа I типа J. Инструкции типа R используют три регистровых операнда. Инструкции типа I используют два регистровых операнда и 16-битную константу. Инструкции типа J (jump) используют 26-битную константой.

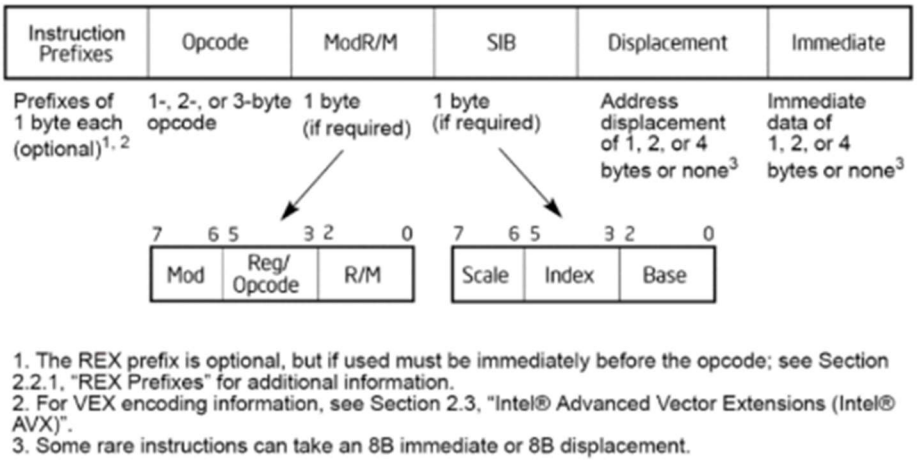


Операция, выполняемая командой, закодирована двумя полями, отмеченными синим цветом: полем op (opcode или код операции) и полем funct (функция). У всех команд типа R поле opcode равно нулю. Операнды закодированы тремя полями: rs, rt и rd. Поля содержат номера регистров. Регистры rs и rt являются регистрами-источниками, а rd – регистром-назначением. Поле, shamt, используется только для операций сдвига.



Инструкции типа I используют в качестве операндов два регистра и один непосредственный операнд (константу)

Этот формат используется только для инструкций безусловного перехода и ветвления. 6-бит поле кода операции (opcode). Оставшиеся биты используются для указания адреса перехода (addr).



1. The REX prefix is optional, but if used must be immediately before the opcode; see Section 2.2.1, "REX Prefixes" for additional information.
2. For VEX encoding information, see Section 2.3, "Intel® Advanced Vector Extensions (Intel® AVX)".
3. Some rare instructions can take an 8B immediate or 8B displacement.

Главная особенность иструкций x86 – не фиксированная длина инструкций что усложняет архитектуру процессора.

Общий вид иснтрукции x86:

- Префикс инструкции
- Опкод (от 1 до 3 байтов)
- ModR/M - определяет саму операцию
- SIB - байт для операций расширенных(32/64 битных)
- Отклонение - используется для вычисления эффективного адреса инструкции(может быть 1 2 4 байта, его размер определяет поле ModR/M)
- Поле непосредственного операнда имеется в командах, использующих непосредственную адресацию; его наличие определяется по коду операции. Размер этого поля обычно равен размеру операндов данной команды; исключением является 64-разрядный режим

6. Типы инструкций. Примеры

<u>Передача данных</u>

- LD и ST (загрузка и сохранение данных из/в память)
- MFC1 и MTC1 (чтение и загрузка в регистры сопроцессора 1)
- MFC0 и MTC0 (чтение и загрузка в регистры сопроцессора 0)

Типы: инструкции загрузки и сохранения данных из/в память, инструкции загрузки констант в регистры, безусловные инструкции пересылки между регистрами процессора, условные инструкции пересылки.

Арифметико-логические
- ADD и SUB
- AND, OR и XOR
- MUL и DIV
- SLT (Set on Less Than) и LUI (Load Upper Immediate)
- двухадресные (Operand1 = Operand1 OPERATION Operand2) и трехадресные (Operand3 = Operand1 OPERATION Operand2)

Управление потоком исполнения
- BEQZ (Branch if Equal to Zero) – условный переход
- JR (Jump Register) и JAL (Jump and Link) – безусловный переход
- TRAP (генерация програм. исключения) и ERET (Exception Return)

Функции:
- вызов процедур
- возврат из процедуры
- организация циклов
- условное ветвление
- безусловный переход

Для работы с плавающей запятой
- ADD.D (Add Double-Precision) и SUB.S (Subtract Single-Precision)
- MUL.D
- C.LT.D (Compare Less Than, Double-Precision) и CVT.S.W (Convert Word to Single-Precision)

Мультимедийные (SIMD)
- ADD.PS и SUB.PS, MUL.PS, C.LT.PS
- .PS (Paired Single) работают с парами чисел с плавающей точкой одинарной точности в одном регистре

Строковые
- REP (Repeat) MOVSB – копирует последовательность байтов из одной области памяти в другую
- REP MOVSW – копирует последовательность 16-битных слов
- REP MOVSD – копирует последовательность 32-битных слов
- обработка последовательностей данных (строк) в памяти без необходимости написания сложных циклов

Системные
- вызовы ОС
- для работы с виртуальной памятью

Для работы с графикой

- операции для работы с пикселями и вертексами
- операции сжатия/восстановления

7. Арифметические инструкции. Инструкции вещественного сопроцессора

1. Типы арифметических операций: сложение, вычитание, умножение, деление.

2. Тип арифметики: циклическая или с насыщением (обработка переполнений путем их циклического сброс или насыщения), знаковая или без знаковая (учитывается или нет знак числа).

Количество операндов

— двухадресные

◦ Operand1 = Operand1 OPERATION Operand2

— трехадресные

◦ Operand3 = Operand1 OPERATION Operand2

Разрядность операндов

— для современных 64-ех битных архитектур составляет обычно 32 и 64 бита

— исключение x86/x86-64 (может использовать меньшую разрядность. Например, 8 или 16)

Генерация исключений и модификация флагов состояния процессора (Арифметические инструкции могут изменять флаги процессора (например, Zero, Carry, Overflow))

Инструкции умножения и деления. Проблема операндов.

Как сохранить удвоенный результат умножения.

1. Результат сохраняется в регистр удвоенной размерности. (например, 128 бит для 64-битных операндов).

2. Старшая часть отбрасывается.

3. Результат сохраняется в два регистра заданных неявно (универсальные или специальные)

4. Для вычисления старшей части операции используется отдельная инструкция.

Деление.

Для деления процессор может возвращать два результата:

1.Только частное: Часто в быстродействующих системах.

2.Частное и остаток: Сохранение в два регистра, задаваемых неявно.

Инструкции вещественного сопроцессора.

– Инструкции загрузки и сохранения.

• Преобразование входных данных ко внутреннему типу.

– Арифметические инструкции.

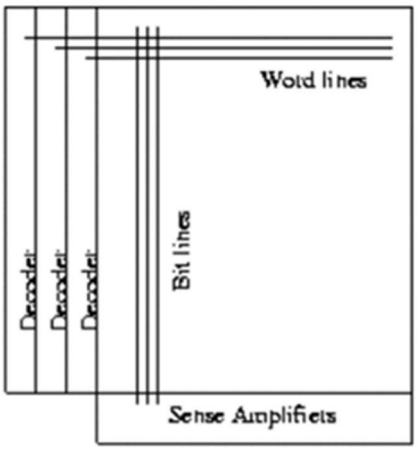
• Mull and Add
• Обратное значение.
• Сложные мат. функции (синус, косинус…).

– Инструкции преобразования типов.
– Инструкции сравнения.
– Результат сравнения помещается во флаги сопроцессора.

8. Регистровый файл. Разновидности регистрового файла

Регистровый файл – модуль микропроцессора (CPU), содержащий в себе реализацию регистров процессора. Основные функции: хранение данных, быстрый доступ к ним и управление процессами чтения и записи. Использование регистровых файлов позволяет выполнять внеочередное исполнение машинных инструкций (не по порядку следования в коде, а по их готовности) и переименование регистров для реализации параллелизма на уровне команд.

Реализован как массив запоминающих ячеек, считываемых вертикально.



Слова (данные) расположены в горизонтальных строках (word lines), и при чтении ячейки выдают своё значение на вертикальные битовые линии (Bit lines). В нижней части эти линии подключены к усилителям, которые преобразуют сигналы с ячеек в булевые сигналы полной амплитуды. В левой части расположены декодеры, активирующие строку, соответствующую заказанному регистру.

Классический регистровый файл

• Количество регистров: 8 – 32
• Разрядность регистров: 32 или 64 бита

+: быстрый
-: программное сохр. требуемых регистров при вызове процедур -> увеличение задержки при переключении

Стековый регистровый файл

• Используется в архитектуре x86.
• Позволяет неявно обращаться к операндам на вершине стека.
• Используется в архитектуре вещественных сопроцессов SPARC и PA-RISC

Оконный регистровый файл

• Используется в архитектуре SPARC.
• Большое количество регистров: до 640
• Доступ к регистрам осуществляется через окно фиксированного размера: 32 регистра.
• Окно разбито на 4 части: Глобальные регистры (неизменны при переключении окон), Локальные регистры (для локальных данных), Собственные входные/выходные параметры, Входные/выходные параметры вызываемой функции.

+: Передача параметров между процедурами происходит через регистры. Можно избежать необходимости сохранять регистры в память
-: Низкая скорость регистрового файла; генерация исключения при переполнении регистрового файла; недостаточная гибкость

Динамический оконный регистровый файл (регистровый стек)

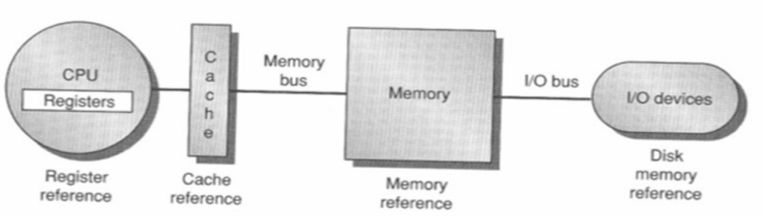
• Используется в архитектуре IA-64
• Регистровое окно произвольного размера
• Количество локальных регистров и регистров входных/выходных параметров произвольно
• Количество глобальных регистров: 32, стековых: 96

9. Организация памяти. Устройство управления памятью (MMU)

Память организована в виде иерархии. Типичная иерархия памяти выглядит следующим образом (снизу вверх):

1. Регистр процессора – самый быстрый, но ограниченный объём памяти, используемый для хранения данных и инструкций, с которыми работает процессор.
2. Кэш-память – быстрый тип памяти, расположенный между процессором и основной памятью. Кэш бывает разных уровней:
    - **L1** – маленький и быстрый, обычно встроен непосредственно в процессор.
    - **L2** – немного больше и медленнее, часто находится рядом с процессором.
    - **L3** – ещё более ёмкий, но медленный по сравнению с L1 и L2, может быть разделён между несколькими процессорами.

3. Оперативная память (RAM) – основная память системы, куда загружаются данные и программы, с которыми работает компьютер в данный момент. Это более медленный, но большой тип памяти.
4. Внешняя память – хранилище данных на длительный срок. Медленнее, чем RAM, но имеет гораздо большую ёмкость.



MMU (Memory Management Unit) – специализированное аппаратное устройство, которое управляет доступом к памяти и отвечает за преобразование виртуальных адресов в физические.

Основные задачи:

1. Преобразование виртуальных адресов в физические для работы с виртуальной памятью, которая позволяет процессам работать с абстракцией памяти, как если бы у каждого процесса был свое изолированное адресное пространство.
2. Разделение памяти между процессами: MMU обеспечивает защиту и изоляцию процессов, чтобы они не имели доступа к ресурсам друг друга.
3. Управление страницами памяти и сегментами: для эффективного использования памяти.
4. Защита памяти: проверка, имеет ли процесс право на доступ к конкретному диапазону памяти.
5. Использование кэш-памяти для ускорения доступа

10. Виртуальная память. TLB

Виртуальная память – концепция, которая позволяет уйти от использования физических адресов, используя вместо них виртуальные.

+: расширение реального адресного пространства; создание изолированных адресных пространств для различных процессов; задание различных свойств для разных участков памяти

Все физическое адресное пространство процесса разбито на виртуальные страницы, вся физическая память разбита на физические страницы такого же размера. У каждого процессора существует таблица, которая связывает виртуальные страницы с физическими страницами. Кроме оперативной памяти страницы могут находиться в файле подкачки

Оперативная память выступает в качестве кэша для «бесконечной» дисковой памяти; для вытеснения страниц из памяти используется LRU-алгоритм (удаление элементов, которые не использовались долгое время); за перенос страниц между диском и ОЗУ отвечает ОС. При отсутствии требуемой

страницы в ОЗУ процессор генерирует исключение. При вытеснении страниц из памяти только модифицированные страницы сохранятся на диск.

TLB (translation look-aside buffer) – это кэш, используемый в устройстве управления памятью (MMU) для ускорения процесса преобразования виртуальных адресов в физические. Он хранит недавние преобразования виртуальных адресов (или страниц) в физические.

Когда процессор обращается к памяти, он генерирует виртуальный адрес. MMU, получив этот виртуальный адрес, должна преобразовать его в физический. Это преобразование осуществляется с использованием таблицы страниц. Для ускорения используется TLB.

На вход TLB приходит виртуальный адрес страницы. Элемент TLB - дескриптор страницы. Процессор содержит два TLB: один для данных, второй для инструкций. Кэш TLB обычно полностью ассоциативный. Может иметь несколько уровней. При переключении процесса TLB очищается.

11. Алгоритм выбора строки жертвы, запись данных в кэш, обработка кэш промаха при записи

Когда кэш заполняется, необходимо выбрать строку для замены. Стратегии выбора строки жертвы:

– Случайная. Удаляется случайная строка из множества.

– FIFO. Удаляется строка, которая дольше всех находиться в кэше.

– LRU. Удаляется строка, к которой реже всего обращаются.

– Замена наименее часто использовавшейся строки (LFU). Заменяется та строка в кэш-памяти, к которой было меньше всего обращений.

Запись в кеш

1 Операции записи встречаются в 5 раз реже операций чтения.

2 Операция записи начинается, только после того, как окончательно выяснено есть нужная строка в кэше или нет.

Две политики записи в кэш:

– Сквозная запись. Одновременно выполняется запись данных как в кэш, так и в основную память.

– Обратная запись. Первоначально запись производится только в кэш. Все кэш линии, в которые происходила запись помечаются специальным флагом. Если такая строка выталкивается из кэша, то производится ее запись в память целиком.

Обработка кэш промаха при записи

1 При использовании сквозной записи в память, строка в кэш памяти не выделяется, а сразу производится запись данных в память.

2 При использовании обратной записи в память:

– В кэше выделяется пустая строка, с расчетом на то, что она будет

полностью заполнена данными.

– Строка загружается из памяти, а затем происходит ее модификация.

12. Кэш: классификация. Пространственная и временная локальность

Разновидности кэша

- инклюзивный (следующий уровень иерархии включает все данные предыдущего уровня)
- эксклюзивный (нет общих данных между уровнями)

По политике управления

- write-through (изменения в кэше сразу записываются в основную память)
- write-back (данные записываются в основную память только при вытеснении из кэша)

По технологии организации

- с прямым отображением direct mapping (каждый блок в основной памяти отображается на конкретную ячейку в кэше)
- полностью ассоциативная fully associative mapping (каждое место в кэше может содержать данные из любой области памяти)
- set associative mapping (гибрид; кэш делится на несколько множеств, и каждый блок памяти может быть размещен в любом из этих множеств, но только в одном месте)

Пространственная и временная локальность программ и данных

- пространственная локальность состоит в том, что если текущее обращение к памяти произошло по адресу «А», то следующее обращение вероятно произойдет по соседнему адресу.
- временная локальность состоит в том, что если произошло обращение к памяти по адресу «А», то вероятно, что в ближайшее время произойдет еще одно обращение по этому адресу

13. Кэш: логическая организация, ассоциативность. Поиск данных в КЭШе

Кэш – высокоскоростная статическая память, расположенная на процессорном кристалле или в процессорном корпусе.

Кэш организуется в несколько уровней:

L1: Самый быстрый и маленький уровень кэша, обычно разделенный на данные и инструкции.

L2: Более объемный, но медленный по сравнению с L1.

L3: Общий для всех ядер процессора, самый объемный, но медленный в иерархии кэшей.

Организация.

Весь объем кэша разбит на блоки или кэш линии. Кроме данных в кэше, для каждой кэш линии хранится ее тэг. Тэг кэш линии хранит старшие биты адреса блока памяти, который в ней содержится.

Ассоциативность.

Соответствие между блоком кэша и блоками памяти определяет понятие *ассоциативности*.

Кэш с прямым отображением устроен так, что каждая строка ячеек основной памяти может отображаться только в одной, фиксированной для неё строке кэша (под строкой понимается минимальная порция информации, которая может передаваться в кэш). Этот тип обладает минимальным временем поиска, но и минимальной вероятностью присутствия необходимой информации в нём.

Полностью ассоциативный кэш, наоборот, позволяет каждой строке основной памяти отображаться в произвольном месте кэша. Этот кэш, наоборот, обеспечивает максимальную вероятность успеха запросов CPU на требуемую информацию, но при максимальном времени поиска.

Множественно-ассоциативный кэш является сочетанием первых двух типов. Кэш этого типа разделён на фиксированное количество областей, именуемое степенью ассоциативности, и каждая строка RAM может отображаться в произвольном месте только одной из областей кэша. И он позволяет добиться оптимальной производительности.

Поиск данных в КЭШе

1) Кэш с прямым отображением.

Этапы поиска в кэше:

1. Извлекается средняя часть адреса (log2m), определяющая номер кэш-линии в кэше.

2. Тэг кэш-линии с данным номером сравнивается со старшей частью адреса (log2N).

Результат:

Если было совпадение по одному из тэгов, то произошло кэш-попадание.

Если не было совпадение ни по одному из тэгов, то произошел кэш-промах.

2) Полностью ассоциативный кэш.

Этапы поиска в кэше:

1. Тэги всех кэш-линий сравниваются со старшей частью адреса одновременно.

Результат, как и в первом случае.

3) Множественно-ассоциативный кэш.

Этапы поиска в кэше:

1. Извлекается средняя часть адреса (log2m), определяющая номер сэта в кэше.

2. Тэги всех кэш-линий данного сета сравниваются со старшей частью адреса (log2N) одновременно.

Результат, как и в первом случае.

14. Кэш: протоколы когерентности

Когерентность кэша – свойство кэшей, означающее целостность данных, хранящихся в локальных кэшах для разделяемого ресурса. Когерентность кэшей – частный случай когерентности памяти.

Когда процессы в системе используют кэширование общих ресурсов, например, памяти, могут возникнуть проблемы с противоречивостью данных. Это особенно справедливо в отношении процессоров в многопроцессорной системе. Когеренция кэшей предназначена для управления такими конфликтами путём поддержания согласованности данных в разных кэшах.

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

- изменение данных в любом из кэшей должно быть распространено на другие копии этих данных в соседних кэшах;
- операции чтения и записи в одну и ту же ячейку памяти должны быть видимы для всех процессоров в одинаковом порядке.

Протокол MSI ,3 состояния,:

Modified: блок изменен в кэше и не соответствует в ОЗУ. Только один кэш в текущий момент может быть в данном состоянии

Shared: блок не изменен и существует как минимум в одном из кэшей в режиме только чтения

Invalid: блок не существует

Протокол MESI

4 состояния:

Exclusive: значение есть только в этом кеше, и оно пока не было изменено

Modified: значение изменено этим процессором, и оно пока не находится ни в главной памяти, ни в кеше какого-либо другого процессора

Shared: значение присутствует в кеше более чем у одного процессора (не строгое состояние)

Invalid: значения нет в кеше

Протокол MESIF

идея: попытаться получить данные от других уровней кэша

проблема: состояние S - сообщения приходят от всех

решение: F(orwarded) – разновидность состояния S – «первый среди равных». Состояние получает всегда последний запросивший данные

Протокол MOSI

Проблема MSI (данные в модифицированноя состоянии):

отмена транзакции -> сброс данных в основную память -> смена состояния на Invalide -> перезапрос копии

Решение: O(wned) – состояние указывает, что данный кэш владеет эксклюзивными данными (отличие от MSI: в памяти данные не актуальны !!!)

MSI / MOSI: состояние O - оптимизация сброса данных в память (write-back)

MSI / MESI: состояние E - оптимизация числа транзакций, необходимых для чтения/записи от другого процессора (ядра)

Протокол MOESI

Modified: блок изменен в кэше и не соответствует в ОЗУ. Только один кэш в текущий момент может быть в данном состоянии

O(wned) – состояние указывает, что данный кэш владеет эксклюзивными данными

E(xclusive)

Shared: блок не изменен и существует как минимум в одном из кэшей в режиме только чтения

Invalid: блок не существует

15. Кэш: способы оптимизации ПО, prefetching

1 Пространственная локальность: Использование данных, которые находятся рядом друг с другом в памяти. Например, считывание массивов последовательными блоками, а не случайными элементами.

2 Временная локальность**:** Повторное использование данных, которые недавно были доступны. Например, использование переменных или структур данных из кэша вместо повторного чтения из основной памяти.

3 Избегать случайного доступа к памяти: Не использовать структуры данных с низкой предсказуемостью доступа, например, хэш-таблицы, если это возможно.

4 Минимизирование кэш-промахов: Использование данных, которые уже находятся в кэше.

Prefetching – это механизм, который загружает данные в кэш заранее, до их фактического использования. Это снижает задержки доступа к памяти.

1 Аппаратный prefetching: Выполняется на уровне процессора. Процессор анализирует паттерны доступа к памяти и автоматически предзагружает данные в кэш. Эффективен для предсказуемых последовательностейдоступа.

2 Программный prefetching: Вызывается из кода с использованием специальных инструкций или методов. Позволяет разработчику явно указать, какие данные нужно загрузить.

- предсказание переходов
  - минимизация hit time
- конвейеризация доступа и многобанковый доступ
  - повышение пропускной способности
  - чаще применяется для кэша инструкций

- неблокирующий кэш
  - применяется для L2/L3
  - пример: выборка инструкций при ожидании получения данных из памяти
- использование write-back кэша
- запрашиваемое слово первым и раннее возобновление выполнения для уменьшения потерь на промахи
  - не дожидаться загрузки всей кэш-линии, передавать данные по мере попадания в кэш
- программная и аппаратная предвыборка данных
  - PREFETCH / PREFETCHA
- компиляторные оптимизации для уменьшения доли промахов

16. Векторные архитектуры. Примеры

Векторизация – это генерация вместо скалярных инструкций векторных инструкций. Векторными называются те инструкции, операндами которых являются вектора. Процесс векторизации может выполняться компилятором или человеком.

Возможности векторизации компилятором ограничены, так как сложно на языке высокого уровня представить большинство векторных операций.

Сложности при разработке векторного процессора: если в коде появляются операции if, многомерные или разреженные матрицы.

Векторные процессоры: Суперкомпьютер NEC SX-Aurora TSUBAS (2017 год, 8 ядер, частота: 1,4 / 1,6 ГГц)

Примеры:

Архитектура x86 (x86-64): MMX, SSE-SSE4, SSSE3, 3DNow!

Архитектура PowerPC: AltiVec, SPE

Архитектура SPARC: VIS

Эти расширения объединяет, то что размер вектора для каждой из них равен 128 бит

2010 Выпуск AVX расширения для x86, размер вектора 256 бит.

17. Векторизация. Векторные инструкции. Развертка циклов

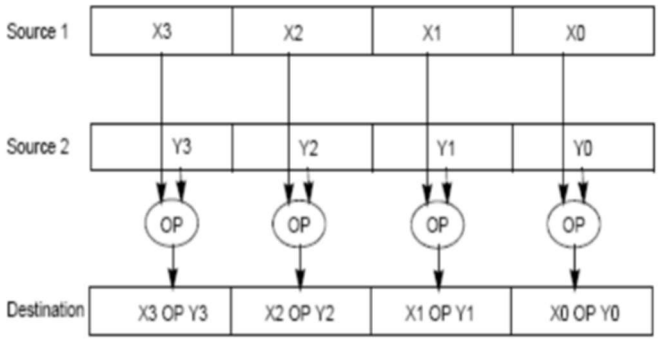
Векторизация – это генерация вместо скалярных инструкций векторных инструкций. Частный случай автоматического распараллеливания кода. Может выполняться компилятором или человеком (компилятором понятно что хуже).

Сложности если в коде появляются операции if, многомерные или разреженные матрицы.

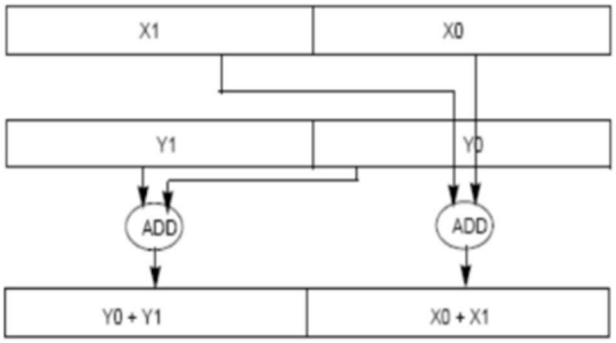
Векторными называются те инструкции, операндами которых являются вектора.

Типы векторных иструкций:

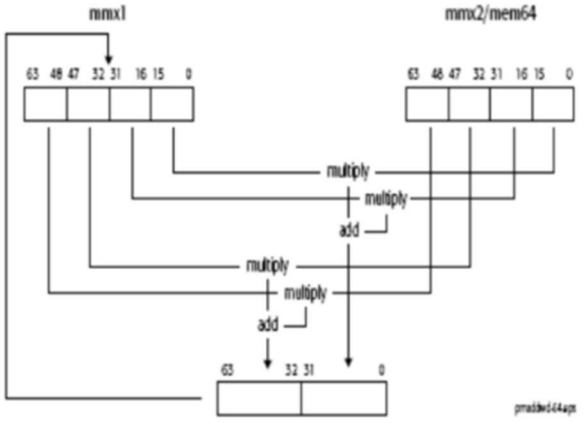
1 Вертикальные - берем условные переменные X0 с 1 источника, Y0 со второго, объединяем их в destination.



2 Горизонтальные – берем условные переменные X1, X0 с 1 источника, Y1, Y0 со второго, в destination записываем как X1 + X0 и Y1 + Y0.



3 Комбинированные – перемножаем n-е элементы с источников, каждые n и n+1 складываем и записываем в destination. Начинаем с n=0.



Развертка циклов – уменьшается количество исполняемых инструкций связанных с вычислением и проверкой условия.

Необходимые условия:

1 цикл for

2 кол-во итераций не должно зависеть от данных

3 неизвестно во время компиляции количества итераций цикла

Недостатки:

1 чем больше инструкций, тем выше вероятность кэш-промаха.

2 может не хватить регистров.

Векторизация выполняется на основе цикла, только для операций, которые внутри цикла. Скалярные элементы объединяются в вектора и скалярные операции заменяются векторными.

Уловия автоматической векторизации:

1 Операция в цикле

2 Итерации цикла не зависимы

3 Число итераций не зависит от данных

4 Внутри цикла не должно быть вызова процедур

5 Шаг между элементами в пямяти - 1.

18. SIMD-инструкции. Примеры

SIMD (Single Instruction stream / Multiple Data stream) – одиночный поток команд и множественный поток данных. Эти системы обычно имеют большое количество процессоров, от 1024 до 16384, которые могут выполнять одну и ту же инструкцию относительно разных данных в жесткой конфигурации. Инструкции SIMD можно найти, в той или иной степени, на большинстве процессоров, в том числе AltiVec фирмы IBM и SPE для PowerPC, MMX, SSE, SSE2, SSE3 SSSE3 и SSE4.x компании Intel.

Целочисленные векторные инструкции входят в наборы MMX, SSE2, SSE3, SSSE3, SSE4.1

В качестве аргументов инструкции могут использовать как 64-ех битные регистры MMX, так и 128-ми битные регистры XMM.

Поддерживается два типа арифметики: с насыщением, без насыщения (циклическая).

Целочисленный векторный набор характеризуется отсутствием ортогональности к типам данных.

Примеры :

1) x86/x86-64:

- MMX, SSE, SSE2, AVX, AVX-512 — работа с 64-, 128-, 256- и 512-битными регистрами.

2) ARM:

- NEON — 128-битные SIMD-инструкции.
- SVE — масштабируемые векторы до 2048 бит.

3) PowerPC:

- AltiVec — 128-битные мультимедиа инструкции.

4) RISC-V:

- RVV — векторы переменной длины.

5) MIPS:
- MSA — SIMD-инструкции с регистрами 128 бит.

19. Конвейер. Характеристики конвейера. Конвейер инструкций

Характеристики:

– Латентность (время выполнения одной команды на конвейере)

– Пропускная способность (время между поступлением двух команд на конвейер)

Не конвейер

– Для быстрого выполнения малого количества операций

Конвейер

– Для быстрого выполнения большого количества повторяющихся операций.

– Время выполнения одной операции больше чем у не конвейерной реализации.

Конвейер инструкций

Стадии:

– Выборка инструкций.
– Выборка данных из регистров и декодирование инструкции.
– Исполнение инструкции (вычисление адреса).
– Обращение к памяти (загрузка и сохранение).
– Запись данных в регистры

Что препятствует эффективному исполнению инструкций на конвейере?

• Зависимости между инструкциями.
• Наличие зависимостей приводит к возникновению конфликтов.

20. Предсказание переходов. Локальный предсказатель

Предсказание переходов является одним из наиболее эффективных способов борьбы с конфликтами по управлению. Идея заключается в том, что еще до момента выполнения команды условного перехода или же сразу после ее поступления делается предположение о наиболее вероятном результате выполнения такой команды (переход произойдет или нет). Последующие команды подаются на конвейер в соответствии с предсказанием.

При ошибочном предсказании конвейер необходимо вернуть к состоянию, с которого началась выборка ненужных команд, т.е. очистить конвейер, что эквивалентно приостановке конвейера. Цена ошибки может быть достаточно высокой, но при правильном предсказании имеем большой выигрыш – конвейер функционирует ритмично, без остановок. Выигрыш тем больше, чем выше точность предсказания.

Точность предсказания – процентное отношение числа правильных предсказаний к их общему количеству.

В самом простом случае рассматривается результат самого последнего выполнения команды. Биты предсказания ветвлений в однобитовой схеме называют переключателем – “принято/не принято”.

В состоянии 0 прогнозируется, что переход не выполняется, в состоянии 1 – выполняемый. Сигналы ПВ (переход выполняется) и ПНВ (переход не выполняется) поступают на вход автомата из блока выполнения команды ветвления. С их помощью осуществляется коррекция прогноза.

Локальный предсказатель – результат тот, который для выдачи результата использует только историю своего перехода

Предсказание ветвления:

Две задачи:

– Предсказание выполнимости перехода.

– Предсказание адреса перехода.

Используется на двух этапах:

– Выборки инструкций.

– Планирование и исполнение инструкций.

Обычно под предсказанием инструкции ветвления понимается предсказание выполнимости перехода, так как относительно заданный адрес перехода легко вычислим.

Предсказатель условного перехода. Два режима работы:

– Предсказание. На вход адрес инструкции, на выходе результат предсказания.

– Корректировка. На вход адрес инструкции и направление перехода.

Результат работы – коррекция внутренних состояний.

Однобитная таблица предсказания ветвления:

• Элемент таблицы: однобитная ячейка.

– Единица – выполнять переход

– Ноль – пропустить переход

– Если предсказание неверно, то инвертировать бит.

• Недостатки:

– Плохо предсказывает поведение коротких циклов. Из-за того, что при выходе из цикла более вероятным становится пропуск перехода.

21. Конфликты исполнения инструкций на конвейере. Типы конфликтов

Конфликты – это некоторая ситуация, при которой невозможно запустить следующую инструкцию непосредственно за предыдущей, не нарушив критерия сохранения корректности программы.

Запуск инструкции происходит спустя n тактов.

Не атрибут программы или алгоритма, это следствие параллельного исполнения зависимых инструкций на процессоре.

Критерии сохранения корректности программы:

1. Сохранение потока данных:

– поток значений передаваемых между инструкциями производителями и потребителями должен быть сохранен;

– инструкции ветвления делают поток данных динамическим.

2. Сохранение поведения исключений

– порядок, количество и тип генерируемых исключений должен сохраниться.

Типы конфликтов:

1. Структурные конфликты – разным инструкциям требуется доступ одному ресурсу процессора.
2. Конфликты по данным – следствие наличия зависимостей по данным именам.
3. Конфликты по управлению – следствие наличия зависимости по управлению.

Способы разрешения конфликтов:

- Программные:
  - обнаружение конфликтов в процессе компиляции и добавление инструкций nop;
  - статическое планирование конвейера.
- Аппаратные:
  - добавление устройства обнаружения конфликтов и останова конвейера;
  - добавление специфичных аппаратных решений.

22. Конфликт по управлению, способы устранения, слот ожидания

Конфликты по управлению возникают при конвейеризации команд переходов и других команд, изменяющих значение счетчика команд. Суть конфликтов этой группы наиболее удобно проиллюстрировать на примере команд условного перехода.

Пусть в программе команда i+1 является командой условного перехода, формирующей адрес следующей команды в зависимости от результата выполнения команды i. Команда i завершит свое выполнение в такте 5. В то же время команда условного перехода уже в такте 3 должна прочитать необходимые ей признаки, чтобы правильно сформировать адрес следующей команды. Если конвейер имеет большую глубину (например, 20 ступеней), то промежуток времени между формированием признака результата и тактом, где он анализируется, может быть еще большим. В инженерных задачах примерно каждая шестая команда является командой условного перехода, поэтому приостановки конвейера при выполнении команд переходов до определения истинного направления перехода существенно скажутся на производительности процессора.

Наиболее эффективным методом снижения потерь от конфликтов по управлению служит предсказание переходов или вставка nop инструкций (менее эффективна). Если после формирования анализируемых признаков оказалось, что направление перехода выбрано верно, все полученные результаты переписываются из буфера по месту назначения, а выполнение программы продолжается в обычном порядке. Если направление перехода предсказано неверно, то буфер результатов очищается. Также очищается и конвейер, содержащий команды, находящиеся на разных этапах обработки, следующие за командой условного перехода. При этом аннулируются результаты всех уже выполненных этапов этих команд. Конвейер начинает загружаться с первой команды другой ветви программы. Так как конвейерная обработка эффективна при большом числе последовательно выполненных команд, то перезагрузка конвейера приводит к значительным потерям производительности.

Если что, слот ожидания - и есть это время, которое мы заполняем с помощью инструкцию nop(no operation - это ассемблерная инструкция)

23. Статическое и динамическое планирование инструкций на примере VLIW архитектуры

VLIW – архитектура, специально разработанная для повышения количества одновременно исполняемых инструкций, позволяет создать более простой процессор, чем RISC архитектура

Статический VLIW процессор

- особенность: за запуск инструкций отвечает компилятор. Компилятор должен правильно расставить nop-ы

  * если инструкция пойдет на исполнение до того момента, как предыдущая инструкция закончится произойдет сбой

  * как быть с устройствами со случайной латентностью (например, устройство загрузки/сохранения при наличие кэша)

  * компилятор всегда выбирает максимальную латентность для устройств со случайной латентностью

недостатки:

- неэффективность кэша
- очень большой размер программы из-за большого количества nop инструкций

Динамический VLIW процессор

Особенность: за запуск инструкций отвечает процессор

- процессор ожидает готовности данных для всех инструкций в пакете.
- только после этого запускает пакет

Позволяет эффективно использовать кэш

Уменьшает размер программы

Общие проблемы VLIW архитектур

• Большой размер скомпилированной программы.

• Отсутствие бинарной совместимости.

- изменение количественного и качественного состава ФУ процессора приводит к бинарной несовместимости с предыдущей версией.

- это приводит к тому, что код нужно перекомпилировать для каждого процессора.

24. Механизмы ускорения выборки инструкций (внеочередное исполнение инструкций, переименование регистров, технологии микро- и макро-fusion)

Внеочередное исполнение

— исполнение машинных инструкций не в порядке следования в машинном коде, а в порядке готовности к выполнению

— основано на алгоритме Томасуло и его развитии

Переименование регистров

— способ реализации параллелизма на уровне команд

• переименование регистров основано на таком же подходе

— между архитектурными регистрами и физическими нет строгого соответствия, при этом физических регистров существенно больше

— при исполнении на CPU архитектурному регистру назначается не постоянная физическая копия

• требуется два файл переименования для связи архитектурных регистров с

физическими

— первый файл содержит информацию о текущих архитектурных регистрах, их

отображении на физические регистры и ячейки БУ

— второй файл содержит информацию о действительных архитектурных регистрах,

их отображении на физические регистры

Технология склейки микроопераций (Micro-fusion)

Объединение в одну микрооперацию несколько зависимых микроопераций при этом результирующая микрооперация должна порождать один результат при внеочередном исполнении нет смысла разделять зависимые операции, т.к. они не могут исполняться параллельно, а всегда будут исполняться последовательно

Наличие нескольких зависимых микроопераций приводит к тому, что станции резервирования и ячейки ROB используются впустую

Объединенная микрооперация будет занимать одну станцию резервирования и одну ячейку ROB

Для поддержки технологии необходимо, чтобы станция резервирования могла хранить дескрипторы для трех и более операндов

После стадии декодирования должно находиться устройство, которое выполняет склейку

25. Спекуляция. Спекулятивный суперскалярный процессор. Основные этапы исполнения инструкций

Сокращения:

- ФУ - функциональное устройство
- СР - станция резервирования
- БУ - буфер упорядочивания
- ССП - спекулятивный суперскалярный процессор

Спекуляция - выполнение некоторой операции, необходимость выполнения которой носит вероятностный характер.

ССП способен выполнять инструкции, которые следуют за невыполненной инструкцией условного перехода, основываясь на данных предсказателя ветвления, и в случае неверного предсказания выполняется переход.

Этапы исполнения инструкции ССП:

1. Планирование инструкции:

   Планирование инструкций осуществляется с вершины очереди планирования.

   Для выполнения планирования должны быть доступны:

   - станция резервирования на требуемом ФУ
   - ячейка в БУ

   Если хотя бы один из ресурсов не доступен, то инструкция ожидает в очереди.

2. Выборка инструкции
3. Ожидание готовности операндов:

   Проверяется существование операндов в регистровом файле. Если они существуют, то записываются в СР.

   Если операндов нет в регистровом файле, то проверяется их существование в БУ. Если операнды есть в БУ, то они записываются в СР.

   Если и в БУ операнды не найдены, то в СР записывается номер ячейки БУ, в которой содержится инструкция для вычисления этого операнда(-ов).

   В файл переименования для результирующего регистра инструкции помещается номер ячейки БУ, которая ее содержит.

4. Исполнение инструкции:

   Инструкции запускаются на исполнении по мере готовности их операндов и доступности ФУ.

По общей шине передается результат инструкции и номер ячейки БУ, которая ее содержит.

5. Запись результата:

Запись результата производится одновременно во все ожидающие СР и БУ.

Инструкция загрузки выполняется в две стадии:

Вычисление адреса.

Чтение из памяти. Запускается если нет конфликтов с предыдущими инструкциями загрузки и сохранения.

Инструкция сохранения на этапе исполнения вычисляет адрес операнда и ожидает готовности результата, запись в память произойдет на стадии завершения.

6. Завершение:

Инструкция после вычисления результата ожидает в БУ своего завершения.

Завершаются инструкции, которые находятся на вершине БУ. БУ поддерживает поведение очереди FIFO.

На этапе завершения:

Для арифметических инструкций и инструкций загрузки проверяется наличия исключений и если исключений нет, то результат записывается регистровый файл.

Для инструкции сохранения проверяется наличия исключений и если исключений нет, то результат записывается в память.

Для инструкции условного перехода проверяется правильно она была предсказана или нет, если да, то инструкция удаляется из БУ, если нет, происходит сброс процессора: очищаются все стадии, на планирование отправляется первая инструкция из правильной ветви.

26. Алгоритм Томасуло. Планирование инструкций

Томасуло - алгоритм динамического планирования инструкций, который позволяет исполнять инструкции в порядке отличном от программного. При одновременном исполнении двух и более инструкций позволяет разрешить RAW (read after write), и устранить WAR (write after read) и WAW (write after write) конфликты/

Достоинста:

- повышение пропускной способности
- уменьшение времени простоя процессора

Недостатки:

- большие аппаратные затраты на реализацию дополнительных устройств

Планирование инструкций:

1 Выборка с вершины ОП (очереди планирования)

- выборка происходит по 1 инструкции за такт

- выборка осуществляется в программном порядке, так как очередь FIFO

2 Декодирование

3 Еазначение на исполнительное устройство

- если все СР устройства заняты, то инструкция возвращается в ОП и ожидает освобождения RS

4 Выборка операндов

- если операнды вычислены, то они выбираются из РФ, если нет то в дескрипторах СР устанавливается ссылки на другие СР

27. Алгоритм Томасуло. Состав процессора. Этапы исполнения инструкций

Состав процессора:

• очередь планирования [ОП]
• регистровый файл [РФ]
• станции резервирования [СР]
• простое вещественное устройство
• сложное вещественное устройство
• общая шина данных [ОШД]
• устройство вычисление адреса
• буфера загрузки
• буфера сохранения
• устройство работы с памятью

Этапы исполнения:

выборка инструкции
планирование инструкции
ожидание готовности операндов
исполнение
сохранение результата

Обработка инструкций загрузки и сохранения

• инструкция загрузки:
  - вычисление адреса
  - выполнение загрузки по адресу
• инструкция сохранения:
  - вычисление адреса
  - ожидания готовности операнда
  - выполнение сохранения по адресу

Ожидание готовности операндов, исполнение, сохранение результатов

• инструкция ожидает в СР до тех пор, пока не будут вычислены все ее операнды и записаны в соответствующие дескрипторы
• передача вычисленного операнда происходит по ОШД вместе с номером СР, который его содержал

• каждая СР слушает ОШД и сравнивает значение номер передаваемого по ней СР с ожидаемым. Если номера совпадает то она забирает значение операнда с ОШД
• передаваемые по ОШД данные сохраняются в регистровый файл
• если все операнды находятся СР, то инструкция отправляется на исполнение

Порядок исполнения инструкций загрузки и сохранения
• определяется наличием зависимостей между инструкциями по ячейкам памяти
• адреса ячеек вычисляются на первом этапе
• инструкция загрузки ожидает завершения всех предшествующих инструкций сохранения по данному адресу
• инструкция сохранения ожидает завершения всех предшествующих инструкций загрузки и сохранения по данному адресу

28. Спекулятивный суперскалярный процессор

Спекуляция – выполнение некоторой операции, необходимость выполнения которой носит вероятностный характер.

ССП - способен выполнять инструкции, которые следуют за невыполненной инструкцией условного перехода, основываясь на данных предсказателя ветвления, и в случае неверного предсказания выполнять откат.

Этапы исполнения инструкции:
1. Выборка инструкции
2. Планирование инструкции
3. Ожидание готовности операндов
4. Исполнение
5. Запись результата
6. Завершения (commit)

Изменения в аппаратной схеме:
- Добавлен буфер упорядочивания (для хранения результатов инструкций между стадией записи результата и стадией завершения) (содержит выполняемую инструкцию и адрес результата инструкции).
- Поля для временного хранения результата инструкции
- Убран буфер сохранения

Ограничения спекуляции:
- Ошибочная спекуляция тратит ресурсы и энергию, НО кол-во инструкций с условным переходом может быть ограничено.
- Инструкция в спекулятивном режиме может вызвать кэш-промах, НО обрабытаются только самые легкие события (например, промах L1), а остальные откладываются до выхода из спекулятивного режима.

29. EPIC. Механизмы поддержки спекуляции

EPIC («*вычисление с явным параллелизмом машинных команд*»)

• Призвана сохранить достоинства VLIW архитектуры и устранить ее недостатки.

• Достоинства VLIW и EPIC: возможность параллельного исполнения инструкций задается на уровне архитектуры, а не определяется динамически в процессе исполнения.

• Недостаток VLIW/ преимущества EPIC: решена проблема бинарной совместимости между различными процессорами.

Проблема бинарной совместимости:

- Программы, скомпилированные для одного процессора, могут не работать оптимально или вовсе не работать на другом процессоре с другим количеством функциональных устройств (ФУ).
- В архитектуре EPIC (включая Itanium) задача назначения инструкций на ФУ перекладывается на процессор, а не на компилятор.
- Процессор сам определяет, какие инструкции запускать на каком устройстве, учитывая своё текущее количество ФУ и их доступность.
- Это означает, что программа не "зависит" от конфигурации конкретного процессора, что обеспечивает бинарную совместимость.

Механизмы поддержки спекуляции:

Спекуляция по управлению

- это возможность корректно исполнять инструкции в ветви, для которой еще не вычислено условие.
- процессоры с архитектурой RISC и CISC выполняют спекуляцию на аппаратном уровне. Первоначально такой возможности в эти архитектуры не закладывалось.
- поддержка спекуляция по управлению включена в состав архитектуры EPIC

Спекуляция по управлению в EPIC основано на:

- наличие двух типов инструкций: спекулятивных и не спекулятивных;
- параллельном исполнении спекулятивных инструкций из двух ветвей;
- использование третьего состояния регистров для сохранения поведения исключений во время спекуляции;

Программная спекуляция по управлению в EPIC:

• достоинства:
  - маленькие аппаратные затраты на реализацию
  - нет затрат на неверное предсказание

• недостатки:
  - требует исполнения большего количества инструкций

- проигрывает по скорости аппаратной спекуляции в случае низкой вероятности неверного предсказания перехода

Спекуляция по данным

Это выполнение инструкции загрузки данных (чтения из памяти) раньше времени, даже если процессор ещё не знает, понадобится ли результат этой загрузки или нет.

Процессор не может просто "перепрыгнуть" через инструкции, которые зависят друг от друга.

Чтобы ускорить выполнение, процессор или компилятор должны попытаться загрузить данные раньше, но при этом нельзя нарушить порядок выполнения: например, нельзя прочитать данные, которые ещё не были записаны.

30. EPIC. Пакет инструкций – способ явного задания параллелизма уровня команд

Пакет инструкций состоит из слотов инструкций и шаблона. Шаблон определяет тип инструкций в слотах и расположение разделителя последовательности инструкций.

Инструкции в последовательности являются независимыми

— запуск инструкций из последовательности начинается, когда конфликты для всех инструкций разрешены

— запускаются процессором в произвольном порядке по мере готовности устройств

— после запуска всех инструкций из последовательности процессор переходи к обработке следующей последовательности

Параллелизм уровня команд – это потенциальное перекрытие нескольких инструкций во времени. Благодаря ILP такие инструкции могут выполняться параллельно, повышая производительность.

Базовый блок – это последовательность инструкций, имеющая одну точку входа и одну точку выхода. Параллелизм присутствует среди инструкций базового блока.

Для получения существенного прироста производительности необходимо использовать параллелизм не только внутри, но и между базовыми блоками.

Простейший способ увеличения ILP – использование параллелизма итераций цикла. Разворачивание циклов – увеличение числа инструкций базового блока (копирование тела цикла несколько раз).

Определение степени зависимости между инструкциями – критический фактор для определения уровня параллелизма в программе. Независимые инструкции могут выполняться одновременно. Зависимые должны выполняться по порядку, однако они могут частично перекрываться во времени.

Зависимость – свойство программы.

Конфликт, возникший (или не возникший) из зависимости – свойство организации архитектуры.

Существует 3 типа зависимостей:

- Зависимости по данным.
- Зависимости по именам.
- Зависимости по управлению.

31. Технология OpenMP

OpenMP – библиотека для параллельного программирования вычислительных систем с общей памятью.

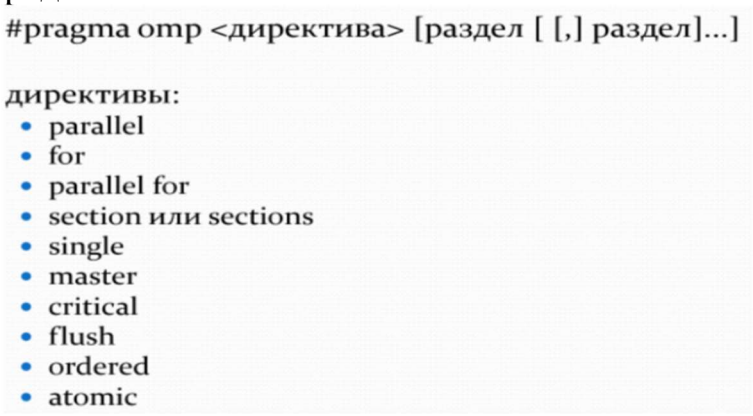
Поддерживаемый языки программирования:

C, C++, Fortran

Поддержка версий:

- актуальная версия – OpenMP 5.2
- MS Visual Studio – OpenMP 2.0 (!!!!) планов по актуализации нет
- GCC 6 – базовая поддержка OpenMP 4.5
- GCC 11 – ожидается полная поддержка OpenMP 4.5 и частичная поддержка OpenMP 5.0

Формат представления:



parallel - распараллеливание потока в отмеченной директивой области видимости.

parallel for - директива распараллеливания цикла

sections – директива omp sections распределяет работу между потоками, привязанными к определенной параллельной области.

Планирование циклов, два основных режима планирования:

статический

- schedule(static)
- schedule(static, 10)

динамический

- schedule(dynamic)
- schedule(dynamic, 10)

1. Тип static:

Если размер не указан, все множество итераций цикла делится на примерно равные части (по числу входящих в них итераций) в соответствии с числом потоков и каждому потоку назначается своя часть. Минимальные затраты на планирование, но возможна несбалансированность нагрузки по потокам.

Размер указан. В этом случае все множество итераций цикла делится на части указанного размера (по числу входящих в них итераций) и полученные части назначаются потокам циклически.

2. Тип dynamic. Части множества итераций (количество итераций в каждой части равно указанному размеру, а если он не указан, то 1 по умолчанию) назначаются каждому потоку по запросу. Т.е., что после начального распределения частей очередная часть достается тому, кто раньше закончил свою работу. Хорошая балансировка нагрузки по потокам, но большие временные затраты на планирование (поскольку операция запроса потока на очередную порцию итераций цикла и его удовлетворения требует относительно больших временных затрат).