Оглавление

Билет 11

Гарвардская архитектура и её отличия от архитектуры фон Неймана. Принципы. Свойства. Особенности и ограничения. Применение на практике. Подходы к обходу ограничений архитектуры. Машинное слово. Control Unit и DataPath.

Принстонская архитектура (фон Неймановская).

Узкое место -- совместная память:
• доступ к инструкциям и данным
• по очереди по одному каналу.

Гарвардская архитектура.

Памяти разделены:

- Память инструкций и память данных — разные устройства,
- Каналы инструкций и данных физически разделены.

Свойства:

- Упрощение выполнения команд.
- Высокая производительность благодаря параллельной обработке.
- Меньшая вероятность ошибок из-за смешивания данных и команд.

Особенности

Плюсы

1. Два физических канала между процессором и памятью.
2. Одновременный доступ к памяти.
3. Разная ширина машинного слова и адреса для данных и программ, вследствие чего получается ооптимизация под решаемую задачу.
4. Меньшая вероятность ошибок из-за смешивания данных и команд.

Минусы

1. Сложность и стоимость реализации.

2. Изоляция инструкций и данных:
    - Запуск результата компиляции.
    - Указатели на функции.

Применения на практике:
Возможно 3 стандартные вариации гарвардской архитектуры:

1. **Архитектура "Память инструкций как данные"** -- реализуется возможность читать и писать данные в память программ. Позволяет генерировать и запускать машинный код.
2. **Архитектура "Память данных как инструкции"** -- реализует возможность запуска инструкций из памяти данных. Позволяет генерировать и запускать машинный код, но параллельный доступ ограничен.
3. **Модифицированная Гарвардская архитектура** -- Доступ к памяти реализуется через независимые кеши для данных и программ, за счет чего, с точки зрения внутренней организации процессора, доступ реализован независимо, при этом канал между процессором и памятью один

Реализации в мире

- Используется в микроконтроллерах и DSP (Digital Signal Processors).
- Микроконтроллеры AVR, ARM Cortex-M.

Подходы для обхода ограничений

- Использование кэш-памяти для команд и данных.
- Оптимизация программного обеспечения для минимизации переключений памяти.

Control Unit — это компонент центрального процессора (ЦП) компьютера, который управляет работой процессора. CU обычно использует двоичный декодер для преобразования закодированных инструкций в сигналы синхронизации и управления, которые управляют работой других блоков.
Datapath — это АЛУ, набор регистров и внутренняя шина(ы) ЦП, которые позволяют данным передаваться между ними.
Машинное слово — Основная единица данных, обрабатываемая процессором, определяющая разрядность архитектуры (например, 32-битное или 64-битное слово).

Билет 12

Механизм микроопераций, микропрограммирование и его роль в развитии компьютерных систем. Особенности и ограничения. Применение на практике. Пример оптимизации через микрокод. NISC архитектура.

Механизм микроопераций -- метод управления операциями в процессоре, при котором каждая инструкция разбивается на более мелкие операции, называемые микрооперациями.

Микропрограммирование -- способ управления выполнением микроопераций путем использования микропрограммы, которая представляет собой последовательность микроопераций для выполнения каждой инструкции.

Роль в развитии компьютерных систем -- Микропрограммирование позволяет создавать более гибкие и модульные процессоры, так как изменение микропрограммы

позволяет изменить поведение процессора без изменения аппаратной части. Это упрощает разработку новых процессоров и обновление существующих.

Достоинства

1. Простота реализации (CISC).
2. Возможность "программирования" системы команд.
3. Доступ к микрокоду для программиста.
4. Генерация ISA под задачу (сократить объём, повысить эффективность)

Недостатки

1. Хранение микрокода в процессоре.
2. CISC долго учить.
3. Разнообразие архитектур →→ проблемы инструментария.
4. Разнообразие команд (форматы, размеры, длительности, доступ). Усложняет:
    - оптимизацию процессора;
    - инструментарий.
5. Микрокод привносит все проблемы программирования (сложность, отладка, методы)

Применение на практике - Использование микропрограммирования позволило создать более сложные и мощные процессоры, такие как серии IBM System/360 и более поздние CISC-процессоры.(**NISC процессора**)

Оптимизация через микрокод - это процесс оптимизации работы процессора путем изменения или добавления микрокодовых инструкций. Микрокод представляет собой набор инструкций, которые используются для управления внутренними операциями процессора.

Пример оптимизации через микрокод может включать в себя добавление новой микрокодовой инструкции для улучшения производительности или исправления ошибки в работе процессора. Например, если обнаруживается, что определенная последовательность инструкций выполняется неэффективно, разработчики могут создать новую микрокодовую инструкцию, которая будет выполнять эту последовательность более эффективно.

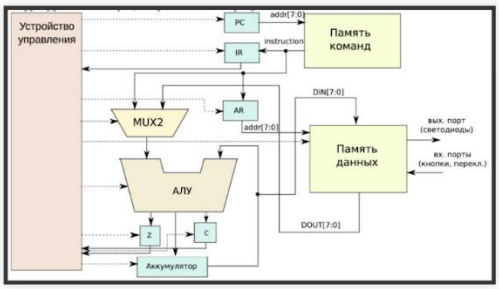
Другой пример оптимизации через микрокод - это исправление ошибок без необходимости выпуска нового процессора. Если обнаруживается ошибка в работе определенной инструкции, разработчики могут создать новый микрокод для исправления этой ошибки и распространить его в виде обновления микрокода (**microcode update**).

Таким образом, оптимизация через микрокод позволяет разработчикам улучшать производительность и исправлять ошибки в работе процессора без необходимости выпуска новых физических версий процессора.

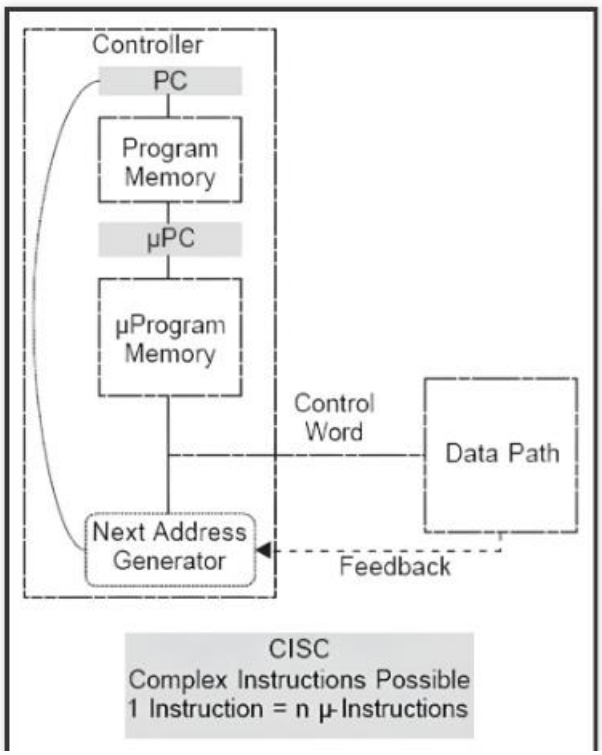
Микропрограммное управление. Пример /1

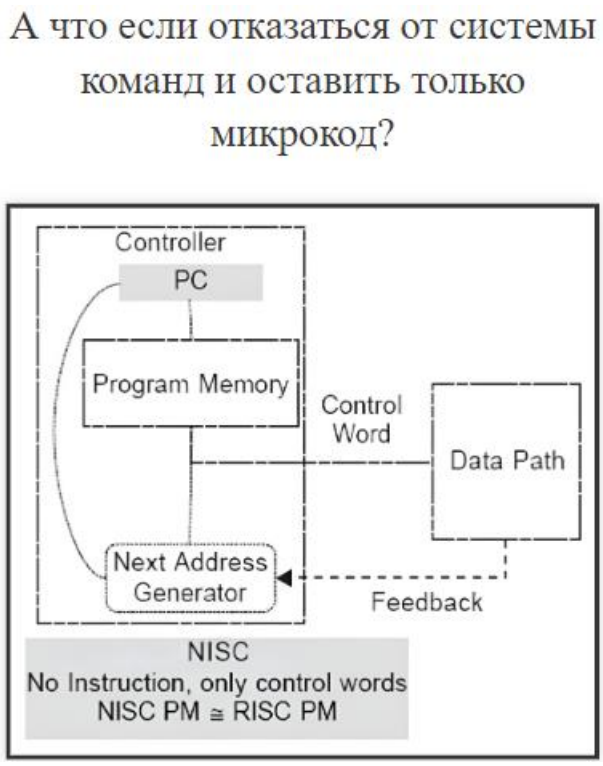
1. Выделяем сигналы управления:
   - защёлкивание регистров;
   - чтение/запись в память;
   - селекторы мультиплексоров.
2. Создаём память микрокоманд. Ширина слова соответствует количеству сигналов.
3. Определяем микрокоманду чтения инструкции (`00-01`).
4. Создаём декодер кода операции в адрес микрокоманд.
5. Реализуем операции в рамках микрокоманд (`n1-n4`), завершая их `micro_jmp 00`.

```
           00 01  ...  n1 n2 n3 n4
latch_PC    1  0  ...   1  1  1  0
mux_PC     +1  0  ...  +1 +1 +1  0

latch_IR    1  0  ...   0  0  0  0
latch_AR    0  0  ...   0  1  1  0

MUX2        0  0  ...  PM  0 DM  0
ALU         0  0  ...  +0  0 +A  0

OE          0  0  ...   0  0  1  0
WR          0  0  ...   0  0  0  1

latch_mPC   1  1  ...   1  1  1  1
mux_mPC    +1 DC  ...  +1 +1 +1 =0
             |
             +-- mpc ← decoder(IR)
```



NISC (No Instruction Set Computer): Концепция, в которой отсутствует явный набор инструкций. Вместо этого управление осуществляется непосредственно через микропрограммы, что упрощает процессор и позволяет гибко адаптировать его к различным задачам.





Достоинства

1. Упрощение аппаратуры.
2. Максимальная эффективность программного управления.
3. Нет ISA, нет проблем её проектирования.

Недостатки

1. Невозможность бинарной совместимости.
2. Низкая плотность машинного кода.

Билет 13

Что такое CISC? Роль в развитии компьютерных систем. Применение на практике. Достоинства и недостатки. Отличия от архитектуры фон Неймана. Особенности программирования.

CISC - это компьютерная архитектура, в которой отдельные инструкции могут выполнять несколько операций низкого уровня (загрузка из памяти, арифметическая операция и сохранение в памяти) или способны выполнять многошаговые операции или режимы адресации в рамках отдельных инструкций.(Архитектура процессора, характеризующаяся наличием большого набора сложных инструкций.)

Достоинства

- Низкоуровневые языки.
- Разнообразие архитектур.
- Неразвитость компиляторов.
- Удобство программирования.
- Высокая производительность.
- Минимизация объёма программ.
- Минимизация накладных расходов.

Низкоуровневые языки - Это помогло программистам на ассемблере, так как они смогли писать более простые программы, ведь всегда найдется инструкция, которая выполняет то, что нужно.
Разнообразие архитектур - было куча разных архитектур и разных систем команд, поэтому разработчики хотели создать что-нибудь получше и универсальнее

Недостатки

- Сложная система команд (использование, анализ).
- Сложное устройство процессора и Control Unit.
- Сложно генерировать эффективный машинный код.

Применение на практике - Широко использовалась в первых компьютерных системах, таких как IBM System/360, и продолжает использоваться в современных x86 процессорах.

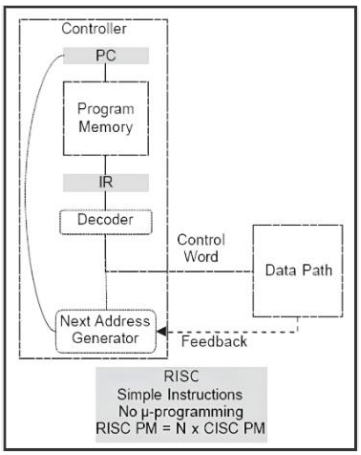
Отличия от архитектуры фон Неймана - CISC имеет более сложную инструкционную архитектуру по сравнению с фон Нейманом, что позволяет сократить количество инструкций, но увеличивает их сложность.(Фон Нейман же делает упор на основные принципы построения компьютеров, включая хранение программ и данных в одной памяти.)

Особенности программирования - CISC упрощает программирование на ассемблере благодаря наличию высокоуровневых инструкций, но требует более сложных

компиляторов. А также оптимизация кода для CISC процессоров может потребовать дополнительных усилий из-за сложности декодирования инструкций.

Билет 14

Что такое RISC? Роль в развитии компьютерных систем. Применение на практике. Достоинства и недостатки. Отличия от архитектуры фон Неймана. Особенности программирования.



RISC -- подход к проектированию процессоров, где быстродействие увеличивается за счёт простого кодирования упрощённого набора инструкций.

Достоинства

- Простота и скорость исполнения инструкций.
- Легкость в разработке и оптимизации процессоров.

Недостатки

- Увеличение объема программного кода.
- Необходимость использования высокопроизводительных компиляторов.
- Ограниченная гибкость: Некоторые задачи могут потребовать большего количества инструкций, чем предусмотрено в RISC-архитектуре.

Роль **RISC** в развитии компьютерных систем она ввела такие идеи как:
1. **Увеличение производительности**: За счет использования простых инструкций RISC-процессоры способны выполнять инструкции быстрее, чем процессоры с более сложными инструкциями.
2. **Уменьшение стоимости** и упрощение конструкции процессора: Простота инструкций позволяет создавать более компактные и эффективные процессоры.

Отличия от архитектуры фон Неймана:
Основное отличие между RISC и архитектурой фон Неймана заключается в количестве и сложности инструкций. RISC использует простые инструкции, в то время как архитектура фон Неймана может использовать более сложные и разнообразные инструкции.

Особенности программирования:
RISC отличается от фон Неймана тем, что фокусируется на уменьшении сложности инструкций в отличии от Фон Неймана, что позволяет повысить производительность за счет увеличения тактовой частоты и параллельного исполнения.

Применение на практике:
RISC-архитектура широко используется в мобильных устройствах, встраиваемых системах, сетевых устройствах и высокопроизводительных вычислительных системах.

Билет 15

Конвейеризированное исполнение команд. Стадии конвейера. Виды конфликтов (по данным, по управлению), их примеры и влияние на производительность. Достоинства и недостатки.

Конвейеризированное исполнение команд - Разбиваем обработку инструкции на несколько этапов и выполняем их параллельно для разных команд.(Один такт — одна стадия конвейера.)

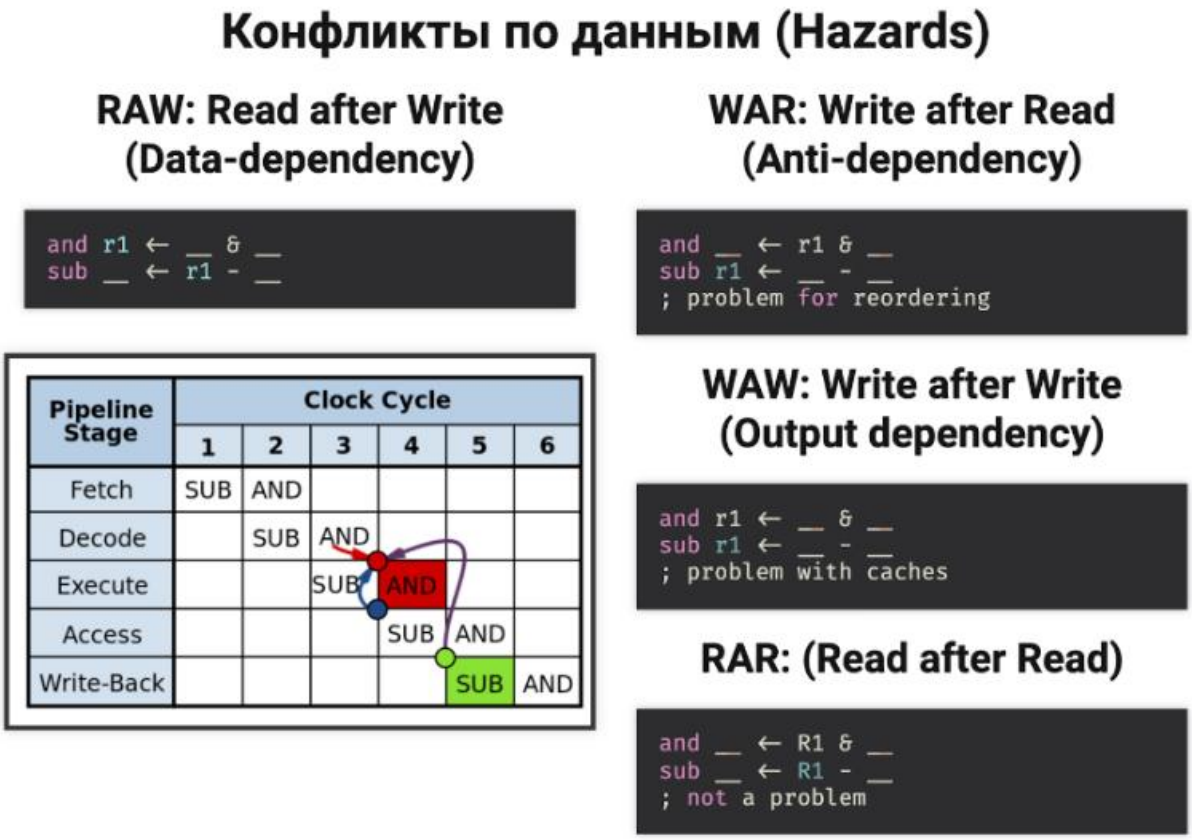
Типовые стадии конвейеров

1. **Instruction Fetch**. Чтение инструкции по счётчику команд.
2. **Instruction Decode**. Декодировать инструкцию и считать регистры.
3. **Instruction Execute**. Операций изменения данных.
4. **Memory Access**. Чтение/запись операндов из памяти/в память.
5. **Write Back**. Запись результата в регистры.

Виды конфликтов

- Структурные конфликты / Structural dependency - Конфликт из-за ресурсов.
  Аппаратура не позволяет выполнить все возможные комбинации инструкций.
  Пример: одновременный доступ к единой памяти команд/данных.
  Варианты полного решения проблемы:
  - Гарвардская архитектура.
  - Двухпортовая память.
- Конфликты по данным / Data Dependency / Data Hazard - Возникают при необходимости использования результатов предыдущих команд, которые еще не завершили выполнение.
- Конфликты по управлению / Control Dependency / Branch Hazards - Конфликт из-за операций условного и/или безусловного перехода.
  Проблема: в конвейер загружены команды, которые не должны быть исполнены.

Данные



Пример: Допустим у нас есть 3 стадии конвейера. Мы считали данные из регистра на 1 этапе. На 2 этапе другая инструкция записала данные в этот регистр. На 3 этапе мы использовали неактуальное значение из п.1 для выполнения 1 инструкции.

RAW - серьезная проблема
WAR - проблема когда меняем команды местами
WAW - редко является проблемой, кажется может влиять на кеши, так как мы делаем две записи в ячейку вместо одной и обязаны обновлять их 2 раза
RAR - не является проблемой

Достоинства

- повышение производительности и уровня утилизации ресурсов.

Недостатки

- снижение скорости исполнения отдельной команды;
- не все операции за один машинный цикл;
- необходимость разрешения конфликтов;
- непредсказуемое время исполнения;
- уязвимости "косвенных каналов" (Meltdown, Spectre)

Не все операции за машинный цикл - деление
Непредсказуемое время исполнения - из-за бренч предикторов
Уязвимости косвенных каналов - произошло обращение памяти, кеш прогрелся, мы получили из другой инструкции доступ к данным, к которым доступа иметь не должны были

Билет 16

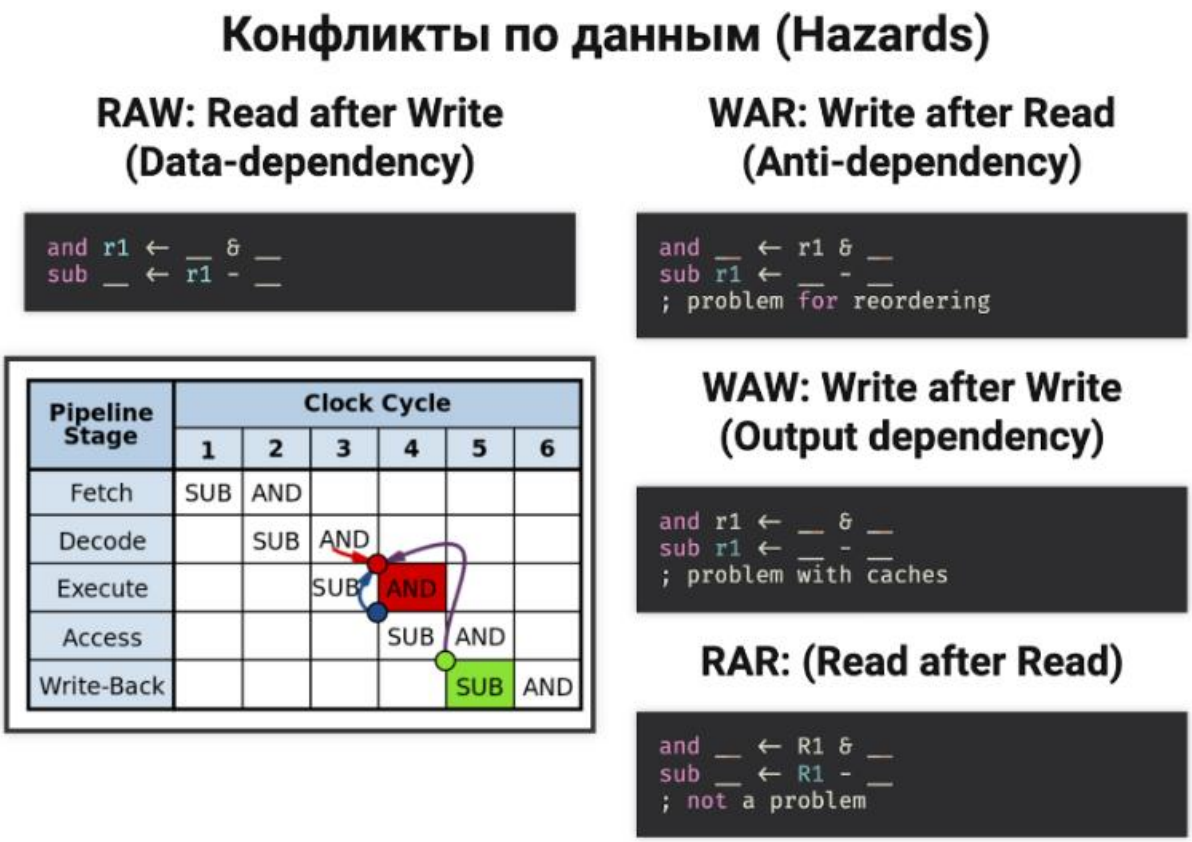
Виды конфликтов при работе конвейера и механизмы их разрешения, сокращения их числа. Пузырёк, разворачивание циклов, предсказания переходов (статические и динамические).

Конвейеризированное исполнение команд - Разбиваем обработку инструкции на несколько этапов и выполняем их параллельно для разных команд.(Один такт — одна стадия конвейера.)

Виды конфликтов

- Структурные конфликты / Structural dependency - Конфликт из-за ресурсов.
  Аппаратура не позволяет выполнить все возможные комбинации инструкций.
  Пример: одновременный доступ к единой памяти команд/данных.
  Варианты полного решения проблемы:
  - Гарвардская архитектура.
  - Двухпортовая память.
- Конфликты по данным / Data Dependency / Data Hazard - Возникают при необходимости использования результатов предыдущих команд, которые еще не завершили выполнение.
- Конфликты по управлению / Control Dependency / Branch Hazards - Конфликт из-за операций условного и/или безусловного перехода.
  Проблема: в конвейер загружены команды, которые не должны быть исполнены.

Данные



Пример: Допустим у нас есть 3 стадии конвейера. Мы считали данные из регистра на 1 этапе. На 2 этапе другая инструкция записала данные в этот регистр. На 3 этапе мы использовали неактуальное значение из п.1 для выполнения 1 инструкции.

RAW - серьезная проблема

WAR - проблема когда меняем команды местами

WAW - редко является проблемой, кажется может влиять на кеши, так как мы делаем две записи в ячейку вместо одной и обязаны обновлять их 2 раза

RAR - не является проблемой

Сокращение числа конфликтов(Мои домысли + GPT)

- **Переименование регистров**: Избегает конфликтов по данным за счёт выделения новых физических регистров для каждого использования логических регистров.
- **Использование кэшей**: Уменьшает задержки при доступе к памяти.
- **Альтернативные потоки исполнения**: Подготовка параллельных ветвей для уменьшения задержек при неверных предсказаниях переходов.

Механизмы решения:
Структурные:

- `0` — пустая операция:
  - занимает конвейер;
  - не выполняет никаких действий;
- просто в реализации;
- снижение эффективности: просто

Данные:

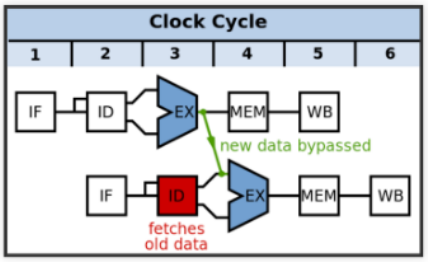
1. Исполнения не по порядку (out-of-order). Компилятор/процессор.

```
i1. R3 <- __ - __      i1. R3 <- __ - __
i2. __ <- R3 + __   => i3. __ <- __ + __
i3. __ <- __ + __      i4. __ <- __ + __
i4. __ <- __ + __      i2. __ <- R3 + __
```

2. **Переименования регистров**. Если зависимость по данным ложная. Запись может быть переназначена на другой регистр (пример WAW).
3. **Вставка пузырька**.
4. **Проброс операндов** (bypassing, operand forwarding) между стадиями процессора, минуя регистровый файл.

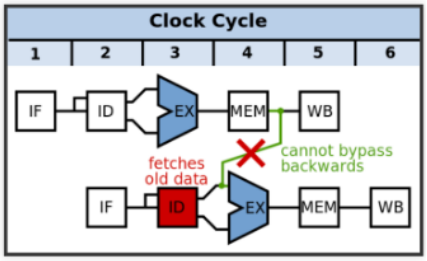
(Другие варианты решения конфликтов)

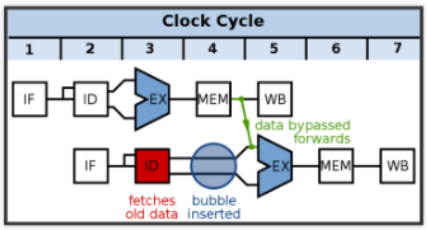
Проброс операндов (Bypassing, Operand Forwarding)



- Запись в регистр `i1:WR`
- Подмена операнда в `i2:EX`
- Проброс значения осуществляется без регистров ⟶ в один такт.

Bypassing + Bubble





Управление:

Разворачивание цикла — это метод преобразования цикла, который помогает оптимизировать время выполнения программы. По сути, мы удаляем или сокращаем итерации. Развертывание цикла увеличивает скорость работы программы за счет исключения инструкций управления циклом и инструкций тестирования цикла.

```
for (int x = 0; x < 100; x++) {
    delete(x);
}
```

↓

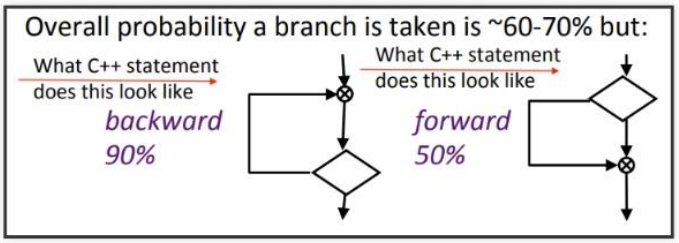
```
for (int x = 0; x < 100; x += 5 ) {
    delete(x);
    delete(x + 1);
    delete(x + 2);
    delete(x + 3);
    delete(x + 4);
}
```

Предсказания переходов:

- **Статические**: Используют фиксированные правила (например, всегда предсказывать переход или непереход).
- **Динамические**: Используют историю выполнения для предсказания переходов. Например, двухбитовые предсказатели или более сложные схемы.

Статические предсказания переходов

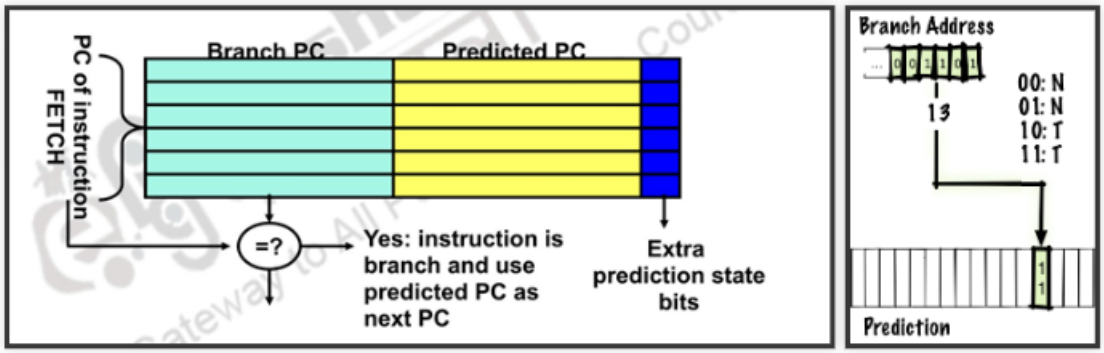
Предсказание определяется инструкцией перехода.



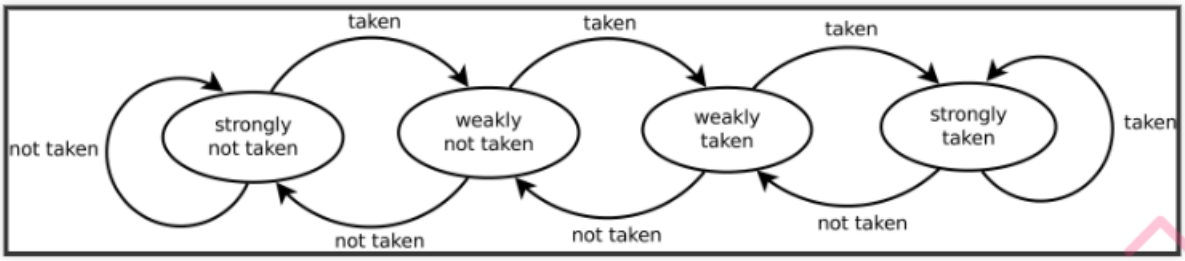
1. Условный переход **вперёд** — не произойдёт.
2. Условный переход **назад** — произойдёт (циклы).
3. Некоторые процессоры (Pentium 4) поддерживают "подсказки компилятора" для предсказаний.

Динамические предсказания переходов

Предсказание использует историю переходов программы.



Счётчик с накоплением, 2-bit



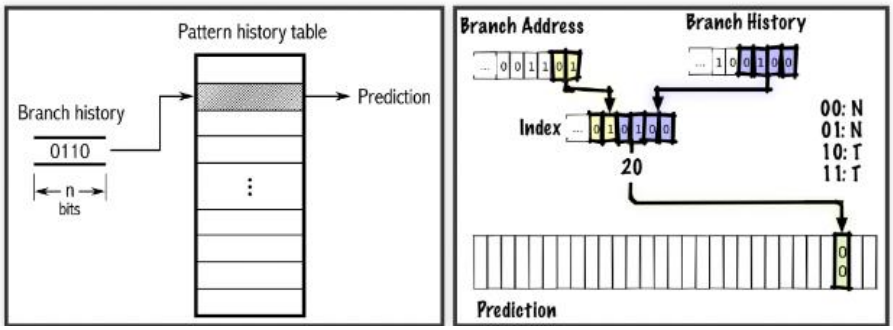
(Усложнённый вариант предсказаний)

Многоуровневые предсказатели переходов

Проблема: у условного перехода есть периодичность (11101110 …).

```
for (int i = 0; i < 3; ++i)
  // code here.
```

- Correlation-Based Branch Predictor
- Branch History — Local (specific branch) or Global (all branches)
- Table sizes. Branch Address collisions. Processes and Threads.
- Больше деталей: https://danluu.com/branch-prediction



Билет 17

Что такое SOP (Stack-Oriented Processors, стековый процессор)? Роль в развитии компьютерных систем. Применение на практике. Достоинства и недостатки. Отличия от архитектуры фон Неймана. Особенности программирования.

SOP (Stack-Oriented Processors) - это тип процессора, который использует стек для хранения и управления данными и инструкциями. В отличие от процессоров с регистровой архитектурой, где данные хранятся в регистрах, в стековом процессоре данные хранятся в стеке, который работает по принципу Last In First Out (LIFO).

- Обработка данных не в регистрах, а на стеке. mul:
  - a = pop(); b = pop()
  - c = a * b; push(c)
- Чтение @ и запись ! из/в память на/из стека.
- Условные и безусловные переходы через стек:
  - если на стеке 0: PC = PC + 1
  - иначе: PC = CONST
  - или иначе: PC = pop()

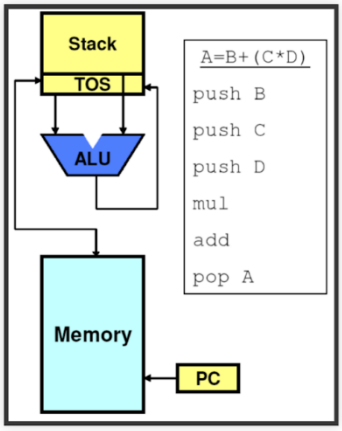
Роль в развитии компьютерных систем:
- SOP имеют свои преимущества в некоторых специализированных областях, таких как встраиваемые системы, где их простота и низкое энергопотребление могут быть выгодными.

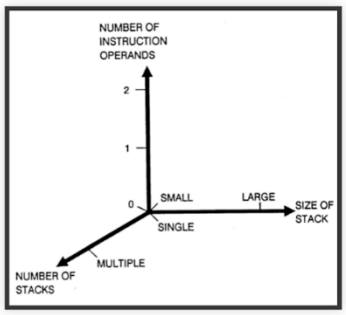
Применение на практике:
- SOP используются во встраиваемых системах, микроконтроллерах, некоторых специализированных вычислительных устройствах.

Stack Machine

ROSC — Reduced Operands Set Computer (скорее шутка)

- Обработка данных не в регистрах, а на стеке. `mul`:
  - `a = pop(); b = pop()`
  - `c = a * b; push(c)`
- Чтение `@` и запись `!` из/в память на/из стека.
- Условные и безусловные переходы через стек:
  - если на стеке `0`: `PC = PC + 1`
  - иначе: `PC = CONST`
  - или иначе: `PC = pop()`





Достоинства

1. High-level language computer architecture.
   - Процедуры.
   - Автоматическая память.
   - Рекурсия.
   - Выражения: X=(A+B)*(C+D)
     →→ A B + C D + *
     →→ A B C D + + *
   - Простой компилятор.
2. Простая система команд, высокая производительность.
3. Cache-friendly.
4. Threads.

Недостатки

1. Эффективность при большом количестве данных на стеке.
2. Динамические структуры.
3. Параллелизм уровня инстр.
4. Сильно отличается.
5. Когда нужно взять несколько значений со стека для одной команды становится тяжело
6. Так как обновлять данные в стеке очень дорого
7. Конфликты по данным будут появляться постоянно

1 - архитектура позволяет писать высокоуровневый код из коробки

2 - команды выполняются за малое кол-во тактов

3 - так как обращения всегда происходят к вершине стека

4 - Чтобы поменять контекст нужно просто переключиться на другой стек, а не ебаться с сохранением и восстановлением данных в регистрах

Отличия от архитектуры фон Неймана:
- **SOP** ориентированы на стековые операции, тогда как архитектура фон Неймана использует регистровый файл и память с адресацией. Данные в Фон Неймане обрабатываются различными устройствами.

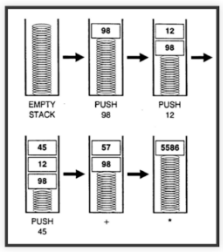
- Регистры заменяются стеком, вершина, которого представляет собой один или несколько регистров. Из-за этого в командах нет операндов, так как это всегда просто поп из стека, из-за этого сильно отличаются системы команд.

Особенности программирования:
- Программирование для стековых процессоров требует использования специальных инструкций для работы со стеком, что может потребовать дополнительной тщательности и внимательности при написании кода.

Forth (стековый язык программирования)

Форт (англ. Forth)
императивный язык программирования на основе стека. Особенности: структурное программирование, отражение (возможность исследовать и изменять структуру программы во время выполнения), последовательное программирование и расширяемость (новые команды).

```
: fac recursive
  dup 1 > IF
    dup 1 - fac *
  else
    drop 1
  endif ;
```



Билет 18

Поддержка операций ввода-вывода в фон Неймановских процессорах. Поддержка на уровне системы команд (порт, отображение в память). Программно-управляемый ввод-вывод. Ввод-вывод через механизм прерываний. Механизм прямого доступа к памяти.

Варианты ввода вывода:

1. **Программно-управляемый ввод-вывод** — операции реализуются процессором. Все действия реализуются через инструкции процессора.(Примеры Работа с ключом; SPI. Echo; Имитация параллелизма.)
   Проблемы:
   - Занимает процессор, включая имитацию таймера.
   - Процессор (алгоритм) должен регистрировать сигнал на частоте в два раза выше частоты сигнала (теорема Котельникова).
   - Высокое энергопотребление.

2. **Ввод-вывод по прерыванию**. Снимает с процессора задачу наблюдения и позволяет это реализовать по внешнему событию.(Устройства генерируют прерывания, чтобы уведомить процессор о необходимости обработки.)
   Источники прерываний:
   - **Аппаратные**.
     - **Внешние** (асинхронные для внутренних циклов процессора): переполнение таймера, нажатие клавиши, сетевой пакет.
     - **Внутренние** (синхронные): деление на 0, ошибка доступа к памяти, и т.п.
   - **Программные** (вызывается инструкцией): взаимодействие программы и ОС.
3. **Channel I/O и прямой доступ к памяти** (Direct Memory Access — DMA). Процессор оповещается об операции завершения ввода-вывода.(Позволяет устройствам ввода-вывода напрямую передавать данные в память, минуя процессор, что значительно увеличивает производительность системы.)
   Плюсы
   1. Скорость.
   2. Эффективность. Снижает нагрузку на процессор.
   3. Параллелизм: DMA может иметь несколько каналов для параллельной работы.
   4. Интерактивность, так как процессор разгружен от "рутины".

   Минусы:
   1. Проблемы совместимости.
   2. Сложность при непоследовательном доступе к памяти.
   3. Ограниченный контроль за системной шиной. Синхронизация работы процессора и DMA.
   4. Конфликты использования DMA разными устройствами ввода-вывода.

   Виды взаимодействия с процессором:
   - **Пакетный режим** (Burst Mode). Приоритет DMA.
   - **Циклический режим** (Cycle stealing mode). Приоритет конфигурируется.
   - **Прозрачный режим** (Transparent Mode). Приоритет процессора.

Варианты ввода с точки зрения ISA:
1. Ввод-вывод через порты. Port-Mapped I/O (PMIO) (Использование специальных инструкций для взаимодействия с периферийными устройствами через порты.)
2. Ввод-вывод через память. Memory-Mapped I/O (MMIO) (Устройства ввода-вывода рассматриваются как ячейки памяти, что позволяет использовать обычные команды чтения и записи.)

Билет 19

Параллелизм уровня задач. Кооперативная многозадачность. Принцип работы и подходы к реализации. Примеры использования. Достоинства и ограничения. Зелёные процессы. Проблема синхронизации процессов по управлению и по данным. Сравнение с альтернативами.

Параллелизм уровня задач - Параллельное выполнение нескольких программ.
Кооперативная многозадачность - Многозадачность, при которой следующая задача выполняется, когда текущая задача явно объявит о готовности отдать процессорное время. Грубо: есть вызов Pause.(Задача сама решает когда передать управления, может хоть не отдавать)

Архитектура Фон Неймана не рассчитана на параллелизм, так как предполагает один непрерывный поток инструкций!!!

Механизмы на которых оно основано:

- Механизм **остановки** выполнения задачи: добровольная передача управления "диспетчеру".
- Механизм **сохранения** состояния задачи: регистры, стек, состояния сопроцессоров, память, ввод-вывод, кеш, состояние предсказателя переходов, и т.п.
- Механизм **планирования** — какой задаче отдать процессорное время следующей.
- Механизм **возобновления** остановленного процесса: восстановление состояния и передача управления.
- Механизмы **изоляции** задач: независимое выполнение, безопасность.
- Механизмы **взаимодействия** между задачами: передача данных и сигналов, общие ресурсы.

Подходы к реализации:

1. Имитация через конечные автоматы (см. прог. ввод-вывод).
2. С диспетчером задач на уровне:
    - ОС (системные вызовы паузы/передачи управления).
    - Virtual Machine/Run Time, "шитый код", time_slice: (Такие конструкции языков программирования как yield, async/await и т.п.)
        - в интерпретаторе,
        - в машинном коде.
    - Программный код(Создается или имитируется программистом):
        - event-loop + callbacks
        - async/await, yield

Может использоваться для:

- Пакетный режим и медленный ввод-вывод (в мейнфреймах), чтобы освободить процессор на время I/O.
- Простые встроенные системы, bare-metal программирование.
- Realtime. Статическое планирование.
- Оптимизации систем, требующих частого переключения задач

Green thread (зеленый поток) - это абстракция потоков выполнения, которая управляется на уровне операционной системы или виртуальной машины и не зависит от нативных потоков операционной системы. (управляется только пространством пользователя)

Проблемы синхронизации процессов

- **Синхронизация по управлению**: Необходимость гарантировать, что задачи правильно передают управление.
- **Синхронизация по данным**: Обеспечение корректного доступа к общим данным.

Например:

- **Deadlock (Взаимная блокировка)**
- **Гонки данных(голодающие философы)**

Существует несколько **альтернативных методов** обеспечения синхронизации процессов:

1. **Мьютексы и семафоры**:
2. **Модель акторов** - В этой модели каждый процесс представлен как актор, который может отправлять и получать сообщения. Это позволяет избежать проблем синхронизации по данным путем изоляции акторов и использования асинхронной коммуникации.
3. **Транзакционная память** - Этот подход позволяет процессам выполнять операции над общими данными в рамках транзакций, гарантируя целостность данных и избегая проблем с гонками данных.
4. **Каналы и сообщения** - Вместо явной синхронизации по управлению или по данным, процессы могут обмениваться сообщениями через каналы, что позволяет им взаимодействовать без явной блокировки или ожидания.

Каждый из этих подходов имеет свои преимущества и недостатки в зависимости от конкретных требований приложения и характеристик системы

Билет 20

Параллелизм уровня задач. Вытесняющая многозадачность. Механизмы переключения задач. Примеры использования. Достоинства и ограничения. Проблема синхронизации процессов по управлению и по данным. Сравнение с альтернативами.

Параллелизм уровня задач - Параллельное выполнение нескольких программ.
Вытесняющая многозадачность - ОС передаёт управление между программами в случае завершения операций ввода-вывода, событий в аппаратуре компьютера, истечения таймеров и квантов времени, поступления сигналов.(В основе вытесняющей многозадачности лежит **система прерываний**.)
Особенности:

- Переключение процессов происходит буквально между любыми двумя инструкциям.
- Распределение процессорного времени осуществляется планировщиком.
- Возможна "мгновенная" реакция на действия пользователя.

Механизмы переключения:

- Механизм **прерывания** процесса — забрать процессор у задачи независимо от её желания.
- Механизм **сохранения** состояния задачи: регистры, стек, состояния сопроцессоров, память, ввод-вывод, кеш, состояние предсказателя переходов, и т.п.
- Механизм **планирования** — какой задаче отдать процессорное время следующей.
- Механизм **возобновления** остановленного процесса: восстановление состояния и передача управления.
- Механизмы **изоляции** задач: независимое выполнение, безопасность.
- Механизмы **взаимодействия** между задачами: передача данных и сигналов, общие ресурсы.

Достоинства:

- Простота разработки ПО с одним процессом.
- Контроль за ресурсами со стороны ОС. Изоляция.
- Интерактивность системы (почти мгновенная реакция)

Недостатки:

- "Лишние" переключения.
- "Тяжесть" процессов (прерывания, состояния, и т.п.).
- Разделяемые состояния и непредсказуемые переключения. Синхронизация. Гонки.
- Непредсказуемая длительность работы. Реальное время.

Пример. Watchdog Timer

Сторожевой таймер — аппаратно реализованная схема контроля от зависания системы. (
Сторожевой таймер - это электронный или программный таймер, который используется для обнаружения и устранения неисправностей компьютера.)
Остальное в предыдущем вопросе.