Микроархитектура процессоров
Задание 3

Вафин Карим, группа М01-407б

23 мая 2025 г.

Introduction

1. Приведите формулы для Performance, Power / Dynamic Power, назовите параметры, входящие в них

- Формула для производительности:

$$\text{Performance} = \frac{1}{\text{Execution Time}} = \frac{1}{t_{cycle} \cdot N_{inst} \cdot CPI} = IPC \cdot f \cdot \frac{1}{N_{inst}}$$

  Параметры:

  - t_{cycle} — длительность одного такта
  - CPI (Cycles per Instruction) — количество тактов на одну инструкцию
  - IPC (Instructions per Cycle) – количество инструкций за такт
  - N_{inst} — количество инструкций
  - f – тактовая частота

- **Динамическая мощность (Dynamic Power):**

$$P_{\text{dynamic}} = \alpha \cdot C \cdot V^2 \cdot f$$

  Параметры:

  - α — коэффициент активности (доля переключающихся транзисторов)
  - C — ёмкость элементов

- V — напряжение питания
- f — тактовая частота

- **Мощность (Power):**

$$P = P_{dynamic} + P_{leak},$$

P_{leak} - потери мощности

2. Объясните законы Мура (Moore's law) и Деннарда (Dennard scaling)

- **Закон Мура (Moore's Law)**: количество транзисторов на интегральной схеме удваивается примерно каждые 2 года.
- **Масштабирование Деннарда (Dennard Scaling)**: при уменьшении размеров транзисторов их частота и плотность увеличиваются, а энергопотребление остаётся примерно постоянным. Это достигается за счёт пропорционального уменьшения напряжения и ёмкости.

3. Когда закончился Dennard scaling? Чем знаменуется его окончание?

Dennard Scaling прекратил действовать примерно в середине 2000-х годов (2005-2006), так как дальнейшее уменьшение напряжения стало невозможным из-за физических ограничений, связанных с утечками тока и нестабильностью работы транзисторов. Это привело к росту плотности мощности и ограничению повышения тактовой частоты, что знаменует окончание эпохи масштабирования производительности с помощью уменьшения размеров.

4. В чем заключается идея Bypassing / Data forwarding оптимизации?

Bypassing / Data Forwarding — это техника, позволяющая избежать задержек, связанных с чтением результатов предыдущих инструкций из регистров. Вместо ожидания завершения записи результата, данные передаются напрямую от одного функционального блока к другому.

Пример: если одна инструкция вычисляет значение регистра, а следующая сразу его использует, процессор может передать результат напрямую в обход записи в регистры или память.

5. Что такое Instruction-Level Parallelism? Приведите примеры оптимизаций для его повышения

Instruction-Level Parallelism (ILP) — это одновременное выполнение нескольких инструкций в рамках одного потока управления.

Примеры оптимизаций для повышения ILP:

- **Pipelining** — разделение выполнения инструкций на стадии, позволяющее обрабатывать несколько инструкций одновременно.
- **Суперскалярность (Superscalar Execution)** – выполнение >1 инструкции за такт.
- **Speculative Execution** — предсказание направления ветвлений и выполнение инструкций до подтверждения.

Out-of-order

1. Для чего нужен Reorder Buffer (ROB)?

ROB используется для поддержки архитектурного порядка инструкций при Out-of-Order Execution. Reorder buffer хранит инструкции и удаляет их, только если все предыдущие инструкциии были выполнены, и при этом обновляется архитектурное состояние процессора.

2. В чем сложность сделать ROB размером 10000 ячеек даже в случае доступной площади и power ресурса?

Размер ROB, помимио площади и мощности, ограничивается следующими причинами:

- При большом размере ROB увеличивается время поиска, обновления и удаления инструкций.
- Branch Mispredictions. При ветвлении в программе процессор должен предсказать дальнейшую ветвь исполнения для дальнешей загрузки команд в ROB. Если случился branch misprediction, то придется тратить ресурсы на очистку ненужных инструкций из буфера.
- Зависимости между инструкциями (Write After Read и Write After Write) понижают эффективность ROB, при этом затраты на поддержание большего буфера увеличиваются

3. Как бороться с False/Anti- Dependencies по регистрам? Что такое Register Aliases Table?

Для борьбы с False Dependencies можно использовать метод **register renaming**, который устраняет ложные зависимости путём назначения уникальных физических регистров для каждой инструкции.

Register Alias Table (RAT) — таблица сопоставлений, которая отображает архитектурные регистры в физические. RAT используется для:

- Назначения нового физического регистра при переименовании
- Отслеживания текущего местоположения значений регистров

4. Для чего нужна Instruction Queue (IQ) (по другому Scheduler Queue (SQ) / Reservation Station)? Когда инструкции удаляются из ROB, а когда из Scheduler Queue?

Instruction Queue хранит инструкции, готовые к исполнению.

Она выбирает инструкции для исполнения, когда все их операнды готовы.

Удаление инструкций:

- Из **Instruction Queue** — происходит при отправке инструкции на исполнение.
- Из **ROB** — происходит при завершении исполнения всех предшествующих инструкций.

5. Что такое Memory Disambiguation проблема? Как с ней бороться?

Проблема Memory Disambiguation заключается в неопределённости порядка между load и store инструкциями при Out-of-Order исполнении

Способы борьбы:

- Анализ адресов — сравнение вычисленных адресов load и store
- Спекулятивная загрузка (load speculation) — выполнение загрузки до выяснения зависимости, но с возможностью отката

- Использование Store Queue — буфера, в котором хранятся ещё не завершённые store, и который проверяется перед выполнением load

6. Объясните смысл разделения Store инструкции на Store Address / Store Data микрооперации

Разделение store-инструкции позволяет выполнять части инструкции независимо и облегчает разрешение зависимостей: для вычисления адреса записи (Store Address) не нужно ждать вычисления данных (Store Data), так как адреса не зависят от записываемых в них значений.

7. Что такое Load speculation оптимизация ?

Load speculation — это оптимизация, при которой инструкции загрузки выполняются до завершения всех предыдущих записей, от которых они могут зависеть. Оптимизация позволяет повысить скорость выполнения инструкции, однако при этом может возникнуть необходимость отката при наличии зависимости между инструкциями.

Branch Prediction

1. Какую информацию хранит Branch Target Buffer?

Branch Target Buffer — это кэш, используемый для ускорения переходов в программах, который хранит следующую информацию:

- Адрес условного или безусловного перехода (instruction PC)
- Целевой адрес перехода (target address)
- Тип перехода (conditional, call, return, etc.)

2. Опишите, как вы бы представили state-of-the-art предсказатель условных переходов на данный момент.

Я бы представил state-of-the-art branch predictor следующим образом:

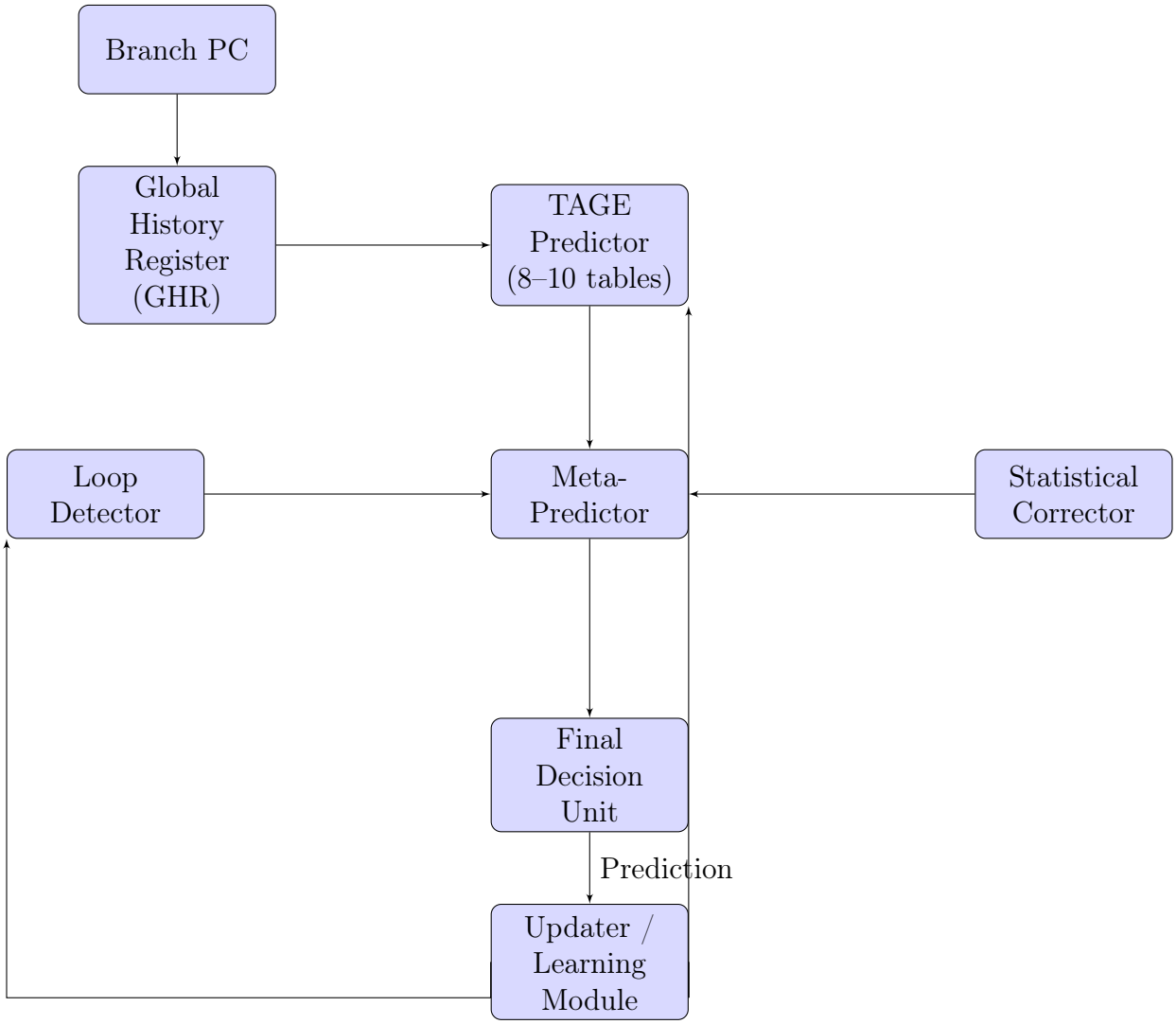

Рис. 1: Архитектура state-of-the-art branch predictor на основе TAGE, Loop Detector и Statistical Corrector

Такой предиктор строится на основе TAGE, и дополняется с помощью Loop Detector и Statistical Corrector. TAGE предиктор является основным модулем, который хранит несколько историй разных размеров. Далее идет Meta Predictor, который анализирует результаты от всех TAGE-компонентов и выбирает наиболее надёжный прогноз на основе контекста. Далее Statistical Corrector, который может быть основан на perceptron, анализирует ошибки основного предсказателя, корректируя прогноз на основе контекста и шаблона промахов. Loop Detector отслеживает повторяющиеся паттерны, и если было обнаружен цикл, то корректирует предсказание соответствующим образом. После предсказания Updater сравнивается с реальным результатом и обновляет счетчики в TAGE, коэффициенты в perceptron и информацию о циклах.
Этот предсказатель был представлен на Competition for Branch Prediction 2023 и достигает точности >95% на большинстве бенчмарков SPEC.

3. Рассчитайте misspredict rate для программы

Для программы:

```
for (int i = 0; i < 100; ++i) std::cout « i « "\n";
```

имеется один условный переход, проверяющий $i < 100$. Он выполняется 101 раз: 100 раз — *Taken*, 1 раз — *Not Taken*.

Распишем таблицу предсказаний

i	**BHR**	**Состояние счетчика**	**Предсказание**	**Результат**
1	000	00 (Strongly NT)	Not Taken	Miss
2	001	00 (Strongly NT)	Not Taken	Miss
3	011	00 (Strongly NT)	Not Taken	Miss
4	111	00 (Strongly NT)	Not Taken	Miss
5	111	01 (Weakly NT)	Not Taken	Miss
6	111	10 (Weakly T)	Taken	Not miss
...	...	...	...	...
100	111	11 (Strongly T)	Taken	Not miss
101	111	11 (Strongly T)	Taken	Miss

Получаем 6 ошибок из 101. Misspredict rate:

$$\frac{6}{101} \approx 5.94\%$$

4. Объясните связь двухуровневых адаптивных предсказателей (two-level adaptive branch predictor) и алгоритма PPM (Prediction by Partial Matching)

PPM (Prediction by Partial Matching) — алгоритм, использующий контекст разной длины для предсказания следующего состояния. Двухуровневые предикторы также используют локальную историю как контекст. Таким образом, обе модели динамически обновляются на основе новых данных.

5. Какими полезными свойствами обладает перцептрон (perceptron) с точки зрения обработки входных данных? Опишите, какую функцию выполняет перцептрон в случае использования в виде статистического корректора (statistical corrector)

Перцептрон хорошо обрабатывает длинные последовательности, а также дополнительно учитывает разные виды данных (признаки), например глобальную и локальную истории переходов и т. д. Перцептрон в виде статистического корректора может комбинироваться с другими предсказателями с целью выявления выбросов и повышения точности предсказания.

6. Какие виды условных переходов являются сложнопредсказываемыми для предсказателя TAGE?

TAGE испытывает сложности при предсказании некоррелированных или слабо коррелированных переходах, а также при присутствии зависимостей, превышающих максимальную длину истории переходов.

7. Объясните основный смысл рассмотренной на лекции статьи “Branch Prediction is Not a Solved Problem”

Основной вывод из статьи: несмотря на множество исследований и разработанных алгоритмов для branch prediction, эта задача остаётся актуальной в виду возможности дальнейшего уменьшения misprediction rate и повышения производительности кода. Основные проблемы заключаются в большом количестве условных переходов, сложных для предсказания.

Необходимы гибридные методы, которые способны детектировать большое множество различных зависимостей при выполнении переходов.

Memory

1. В чем разница между Fully Associative, Direct mapped и Set-Associative кэшами?

- **Fully Associative:** Блок данных может быть размещён в любом месте кэша, поэтому есть необходимость полного сравнения всех тегов ячеек. Коллизий не происходит.
- **Direct Mapped:** Каждый блок памяти отображается в строго определённую ячейку кэша. Быстрый поиск, но возможны коллизии.
- **Set-Associative:** по сути гибрид предыдущих видов. Кэш делится на множества (sets), каждый блок памяти отображается в одно множество, но может быть размещён в любом из его n блоков (n-way associativity).

2. Для чего используется Translation Lookaside Buffer (TLB)?

TLB — это кэш таблицы страниц, используемый для ускорения виртуальной адресации:

- Хранит недавние сопоставления между виртуальными и физическими адресами
- Позволяет избежать длительного обхода иерархии таблиц страниц при каждом обращении к памяти
- Таким образом ускоряется работа MMU (Memory Management Unit)

3. Что такое гранулярность политик замещения?

Гранулярность политик замещения определяет, на каком уровне принимается решение о замене данных в кэш-памяти. Выделяются следующие типы гранулярности:

- **Строковая (line-level)**
- **Блочная**

- **Постраничная**

4. Какие есть стандартные паттерны обращения к кэшу?

- **Scalar/zero-stride** – доступ к одной и той же ячейке, например
  for (int i = 0; i < 10; ++i) { a[0] = 1; }
- **Streaming** – последовательный доступ, например
  for (int i = 0; i < 10; ++i) { a[i] = 1; }
- **Constant stride** – доступ с определенным фиксированным шагом, например
  for (int i = 0; i < 10; i += 2) { a[i] = 1; }
- **Thrashing** – данные вытесняют друг друга, например, обход графа.

5. В каком ключе LRU Insertion Policy (LIP) лучше классической LRU политики замещения?

При классической LRU новые элементы добавляются в начало списка, старые вытесняются при необходимости, а при LRU Insertion Policy (LIP) новые элементы вставляются в конец списка, то есть считаются наименее используемыми. Таким образом LRU Insertion подходит при паттерне thrashing, снижая вытеснение часто используемых данных из кэша.

6. За счет чего BRRIP политика замещения является устойчивой к scanning и thrashing паттернам?

BRRIP устойчива к trashing, так как большинство новых данных вытесняются быстро, если они не были переиспользованы. В случае scanning паттерна также новые данные вставляются не в начало кэша, а ближе к концу, таким образом они не вытесняют более долгоживущие данные.

7. В чем основная идея политик замещения Hawkeye и Mockingjay ?

Основные идеи политик Hawkeye и Mockingjay:

- Использование профилирования, чтобы различать кэш-дружественные и кэш-недружественные блоки.

- Обучение предсказывать, какой блок не будет переиспользован.
- Политика вытеснения имитирует оптимальный алгоритм (Belady's MIN).

8. Объясните принцип работы Spatial Memory Streaming (SMS) префетчера

Принцип работы SMS:

- Обнаруживает шаблоны обращений внутри страниц памяти, например, кэш-линии в пределах одной страницы.
- При загрузке одной линии предсказывает и подгружает связанные линии в том же регионе.
- Адаптируется к изменению паттернов. Если программа меняет способ доступа к данным, SMS прекращает ненужную предвыборку. Это помогает избежать бесполезных загрузок и снижает нагрузку на память.

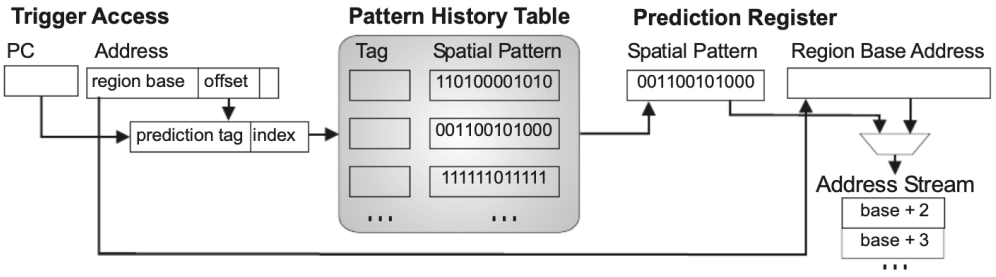

Рис. 2: SMS Prefetcher

Advanced Optimizations and Parallelism

1. На покрытие какие сложных случаев направлено использование Execution-based Prefetchers?

Execution-based Prefetchers нацелены на выявление зависимостей между последовательностями инструкций и адресами памяти, которые невозможно предсказать простыми шаблонами, то есть из-за которых возможны промахи в кэш.

2. Что такое Value Prediction оптимизация? Для каких инструкций, как правило, используется данная оптимизация в современных процессорах?

Value Prediction — это оптимизация, при которой значение результата инструкции предсказывается до её выполнения. Чаще всего используется для арифметических инструкции и инструкций загрузки.

3. За счет чего получается прирост производительности в случае применения Value Prediction оптимизации?

За счет того, что предсказанные значения позволяют запускать зависимые инструкции спекулятивно, не дожидаясь завершения текущей инструкции. Это увеличивает Instruction-Level Parallelism.

4. В чем разница между Value Prediction и Branch Prediction с точки зрения необходимости / ожидаемого положительного эффекта от предсказания и последующего спекулятивного исполнения в случае низкой уверенности в точности предсказания?

- **Branch Prediction:** Ошибочное предсказание может потребовать отката большого количества инструкций.
- **Value Prediction:** Ошибки менее критичны, так как обычно затрагивают только зависимости от одной инструкции; возможна частичная коррекция без полной очистки пайплайна.

Таким образом, точность имеет большее значение для ветвлений, а Value Prediction может использоваться при более низкой уверенности.

5. Что такое Memory Renaming оптимизация?

Memory Renaming — это оптимизация, применяемая для устранения или уменьшения ложных зависимостей по данным в памяти (false dependencies), таких как WAR и WAW.
Эти зависимости возникают, когда разные инструкции обращаются к одним и тем же адресам памяти, но на самом деле не зависят друг от друга.
Memory renaming помогает повысить уровень параллелизма команд.

Обычно процессор при записи в память должен предположить, что любые последующие операции чтения из этого адреса могут зависеть от этой записи. Это ограничивает спекулятивное выполнение. При оптимизации Memory Renaming процессор динамически переназначает адреса памяти на временные наименования, чтобы убрать ложные зависимости между инструкциями, которые работают с одинаковыми адресами, но по факту не связаны.

6. Объясните разницу между Fine-Grained Multithreading, Coarse-Grained Multithreading и Simultaneous Multithreading.

Fine-Grained Multithreading

- Процессор переключает контекст между несколькими потоками на каждом такте.
- Инструкции из разных потоков чередуются по одной инструкции за цикл.
- Используется для маскировки задержек исполнения (например, ожидания данных из памяти).
- Требует наличия аппаратного управления состоянием всех активных потоков.
- Хорошо скрывает задержки доступа к памяти.
- Быстро реагирует на изменение состояния потоков.
- Высокая сложность управления множеством потоков.
- Может снижать производительность каждого отдельного потока из-за частой смены контекста.

Coarse-Grained Multithreading

- Процессор выполняет инструкции одного потока до тех пор, пока не возникнет ситуация, требующая ожидания, после чего происходит переключение на другой поток.
- Меньше аппаратных затрат по сравнению с FGMT.
- Меньше накладных расходов на переключение контекста.
- Проще в реализации.

- Неэффективно скрывает короткие задержки.
- Долгие переключения могут не успевать маскировать некоторые типы задержек.

Simultaneous Multithreading (SMT)

- Инструкции из разных потоков выбираются и исполняются одновременно.
- Эффективно использует простаивающие функциональные блоки.
- Высокая пропускная способность.
- Подходит как для маскировки коротких, так и долгих задержек.
- Требует значительных аппаратных ресурсов.
- Сложность в управлении зависимостями между потоками.
- Возможна конкуренция за общие ресурсы, что может снизить производительность отдельных потоков.