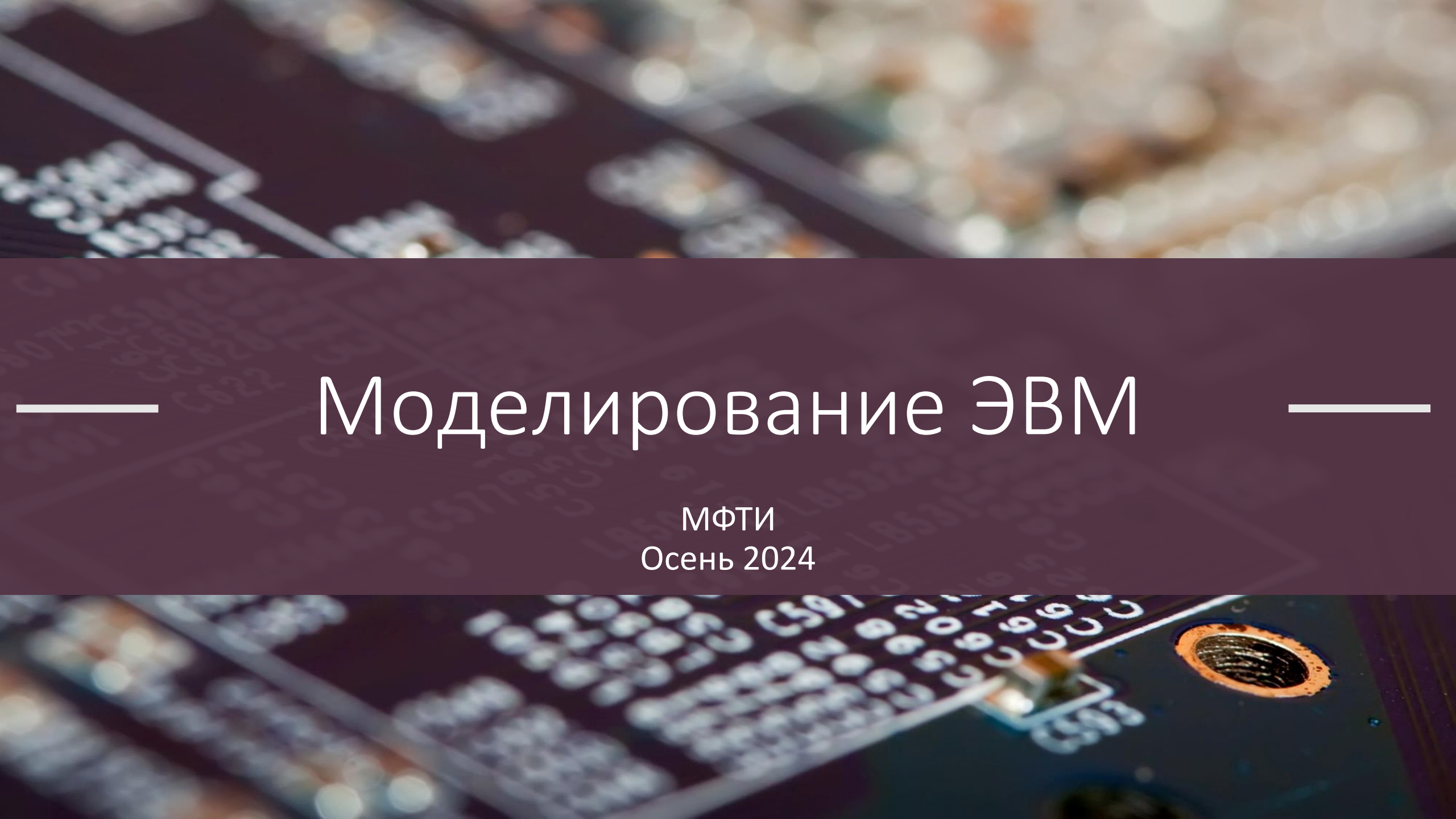


Моделирование ЭВМ

МФТИ
Осень 2024

Классификация симуляции

По детальности симуляции:

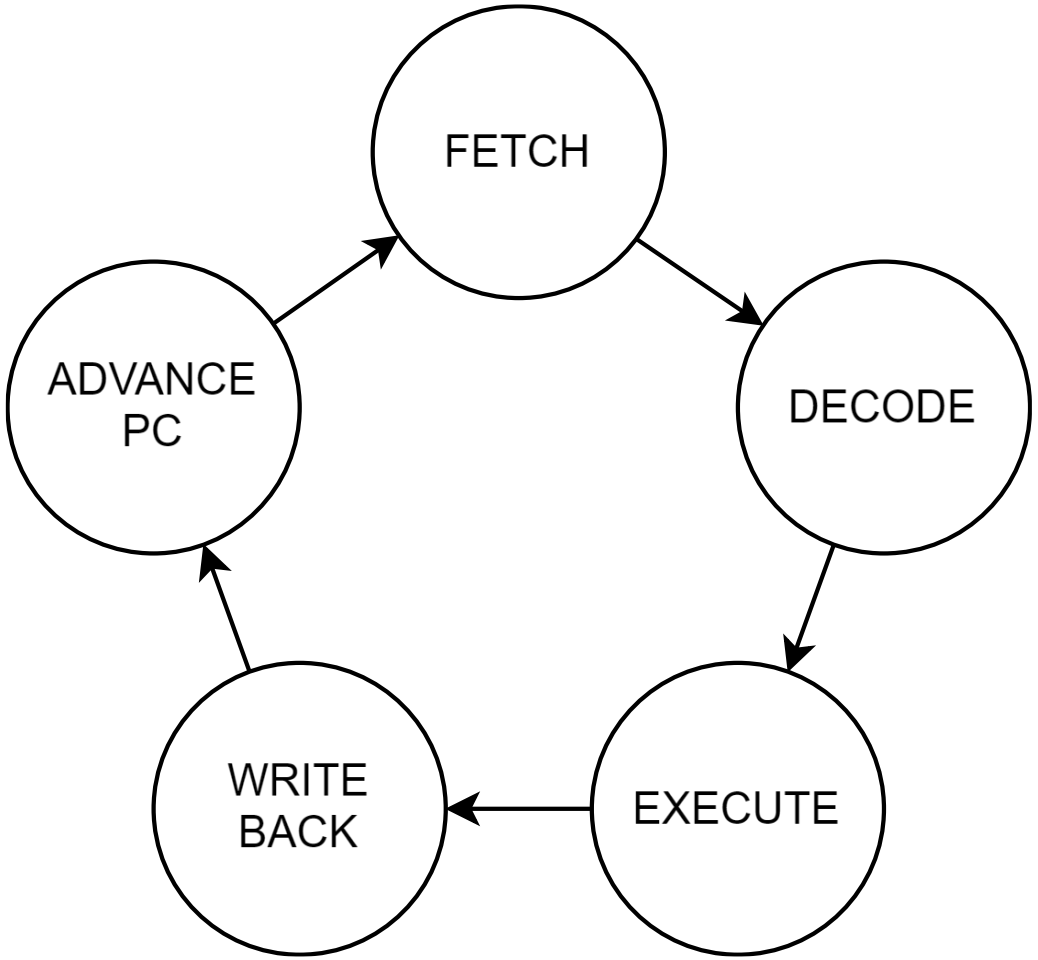
- Функциональная (ISA)
- Микроархитектурная
  - Cycle-approximate
  - Cycle-accurate
- Физическая

Классификация симуляции

По охвату симуляции:

- Модуль
- CPU
- SoC (System on Chip)

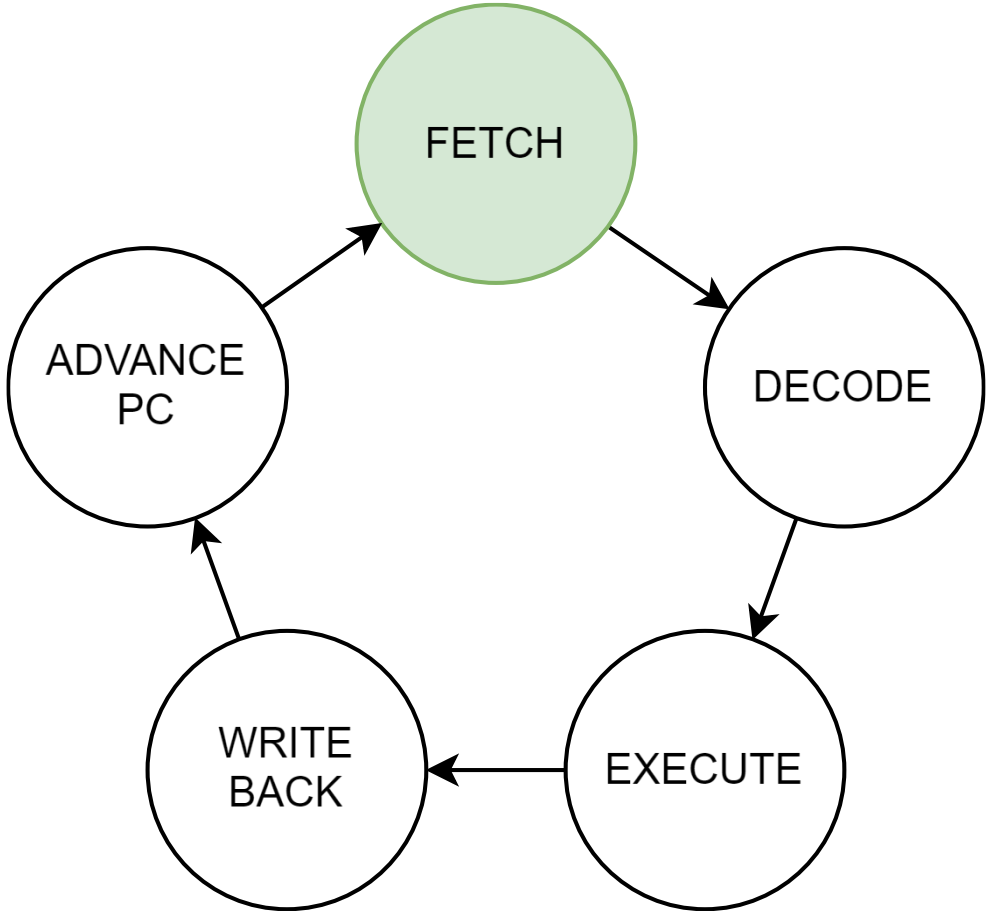
Функциональный симулятор: интерпретация



Функциональный симулятор: интерпретация

Fetch:

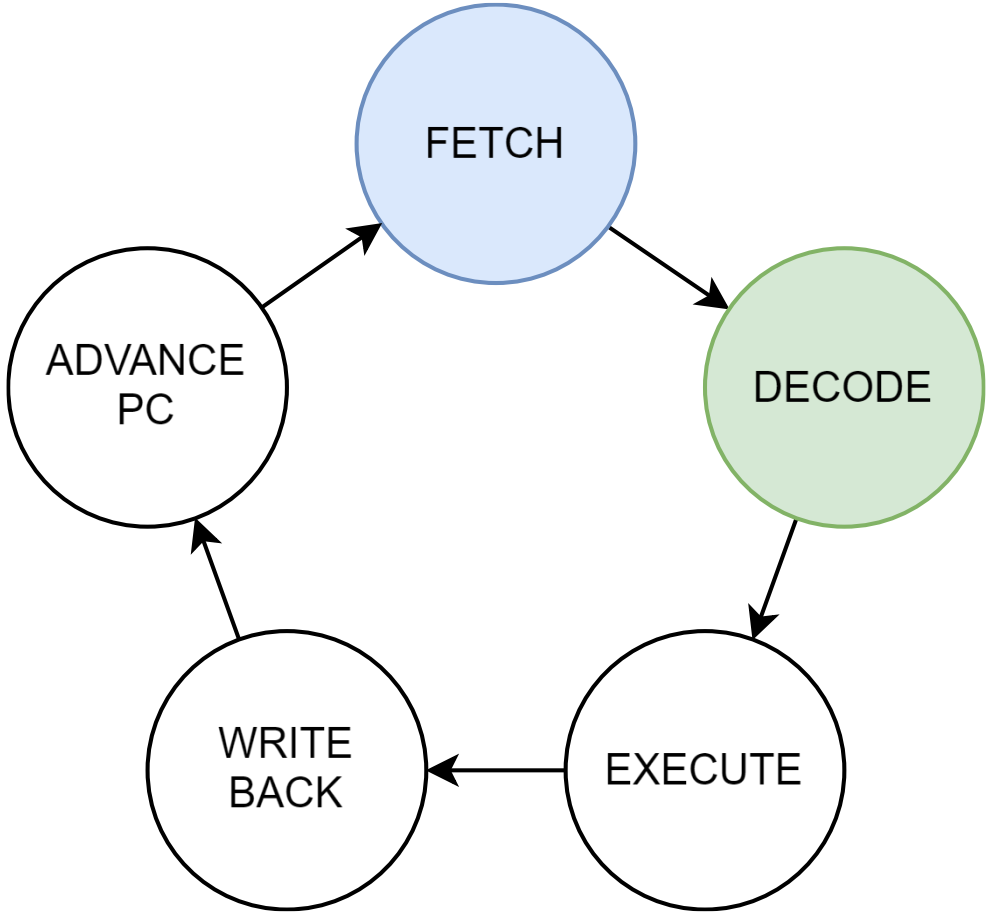
- Загрузка инструкции из памяти



Функциональный симулятор: интерпретация

Decode:

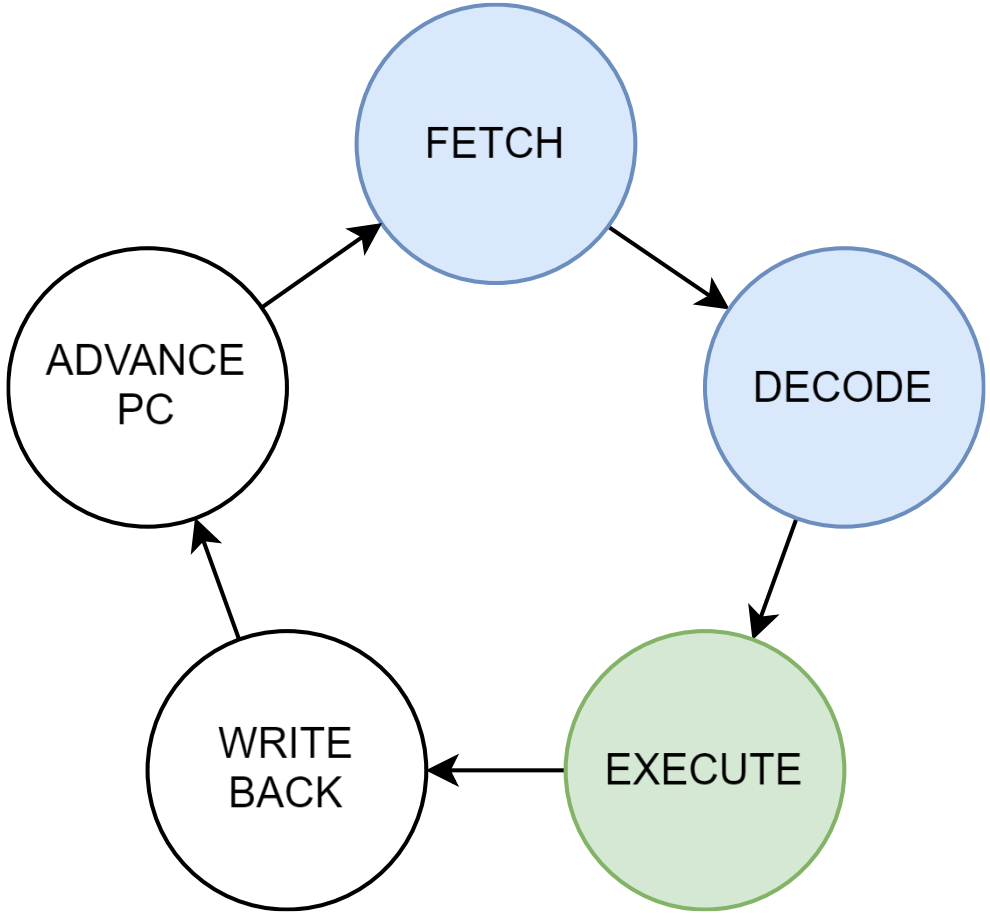
- Декодирование загруженной инструкции



Функциональный симулятор: интерпретация

Execute:

- Выполнение инструкции

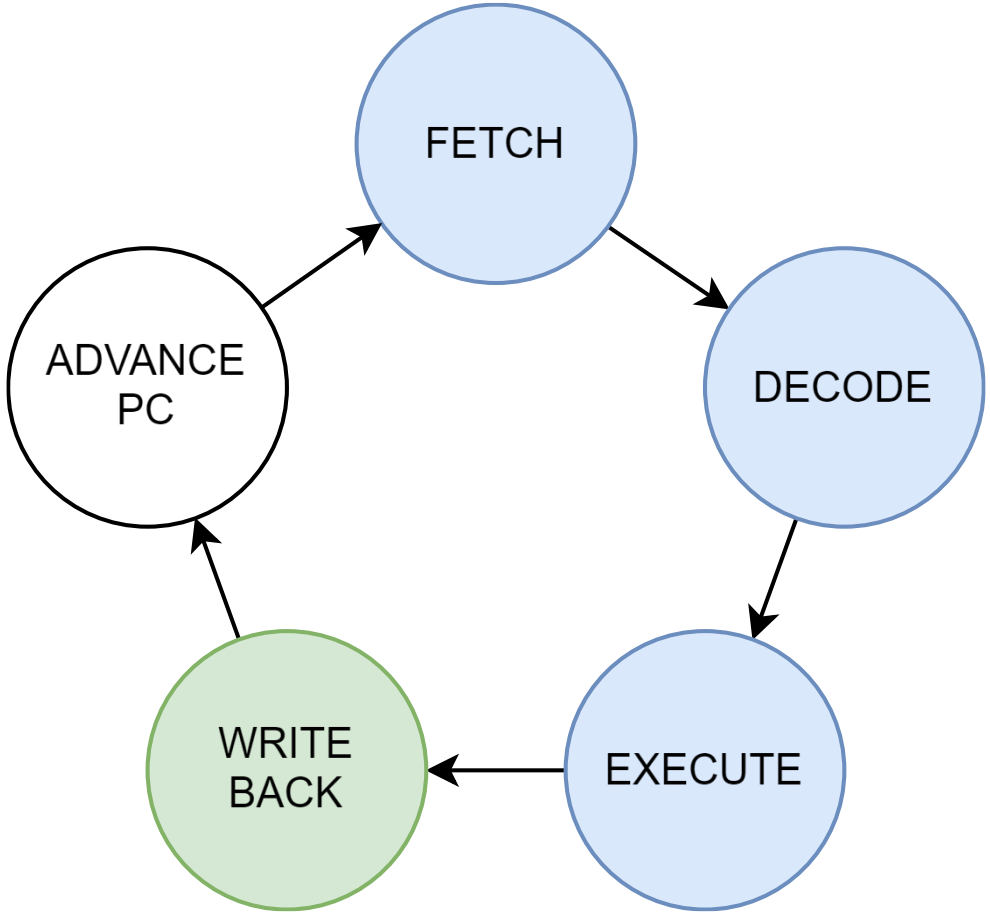


Функциональный симулятор: интерпретация

Write back:

- Запись результатов выполнения инструкции

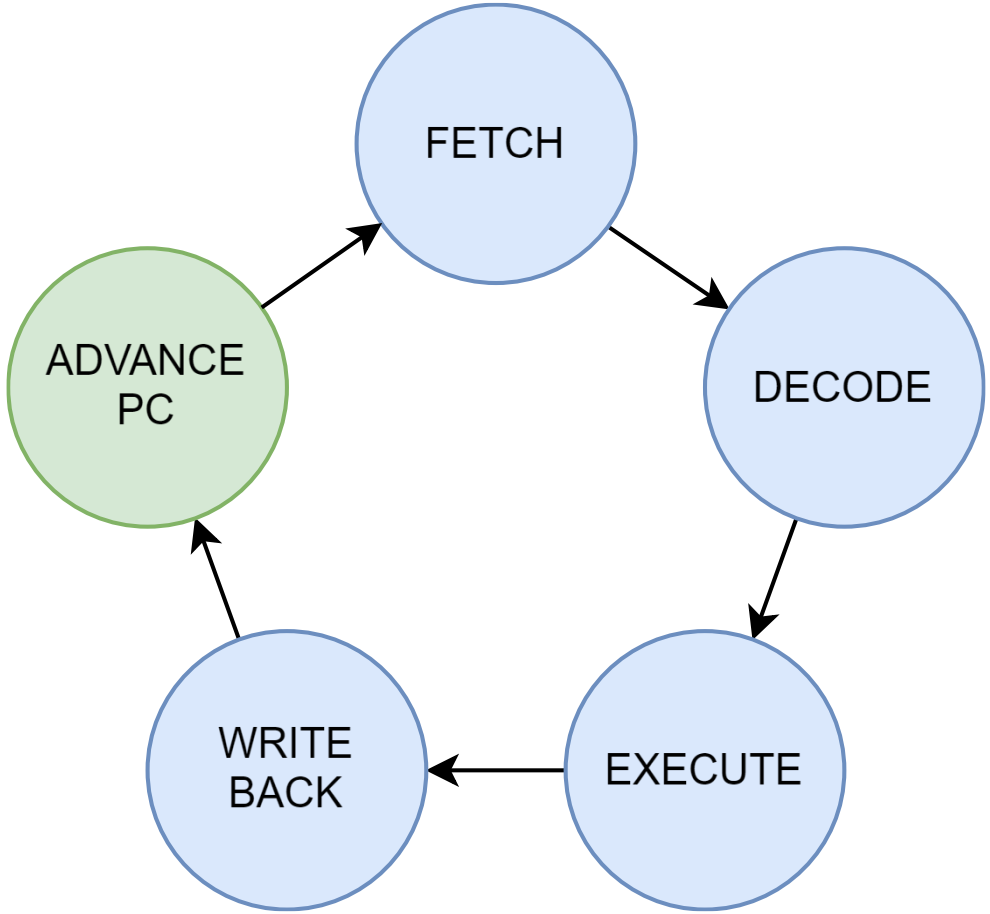
- Эта операция может быть не очень простой, если симулируется платформа с аппаратной трансляцией адресов



Функциональный симулятор: интерпретация

Advance PC:

- Продвижение вперед program counter’a



Функциональные симуляторы RISC-V

- spike (riscv-isa-sim)
  - Самый первый функциональный симулятор RISC-V
  - Работает через интерпретацию
- Sail (sail-riscv)
  - Симулятор, генерируемый из формального описания ISA
  - Работает через интерпретацию
  - Идея шикарная, но никто не захочет писать на таком языке (Ocaml)
- QEMU (Quick EMUlator)
  - Основной симулятор, используемый для раннего прототипирования
  - Самый быстрый функциональный симулятор (скоро узнаем почему)
  - Включает продвинутую поддержку привилегированной части ISA и симуляцию различной периферии (сетевые устройства, жесткий диск, …)

Пробуем запускать функциональные симуляторы

Лабораторная по бенчмаркингу симуляторов:
testgen-lectures/labs/sim-bench

Тестировать производительность симуляторов мы будем на бенчмарке coremark:

- Spike:

```
time spike pk /opt/coremark/coremark.exe
              ^^--- proxy kernel, с его помощью spike печатает
```

- QEMU:

```
time qemu-riscv64 /opt/coremark/coremark.exe
```

Обсуждение: преимущества и недостатки функциональных симуляторов

Что на ваш взгляд выделяет функциональные симуляторы в лучшую сторону?

Есть ли проблемы у функциональных симуляторов?

Преимущества и недостатки функциональных симуляторов

- Преимущества:
  - Быстрее остальных
  - Простота реализации
- Недостатки:
  - Недостаточная детализация симуляции

Недостаточная детализация функциональных симуляторов

Рассмотрим кусочек кода на ассемблере:

```
addi x6, x0, 42
ld   x5, 0x40(x2)
sd   x5, 0x48(x2)
add x10, x7, x0
```

Недостаточная детализация функциональных симуляторов

Рассмотрим кусочек кода на ассемблере:

```
addi x6, x0, 42
ld   x5, 0x40(x2)
sd   x5, 0x48(x2)
add x10, x7, x0
```

Как этот код будет выполнен на функциональном симуляторе:

```
-> x6 = 42
-> x5 = mem(x2 + 0x40)
-> mem(x2 + 0x48) = x5
-> x10 = x7
```

Недостаточная детализация функциональных симуляторов

Рассмотрим кусочек кода на ассемблере:

```
addi x6, x0, 42
ld   x5, 0x40(x2)
sd   x5, 0x48(x2)
add x10, x7, x0
```

Будет ли этот код так же выполнен на реальном процессоре?

Недостаточная детализация функциональных симуляторов

Рассмотрим внимательнее кусочек кода на ассемблере:

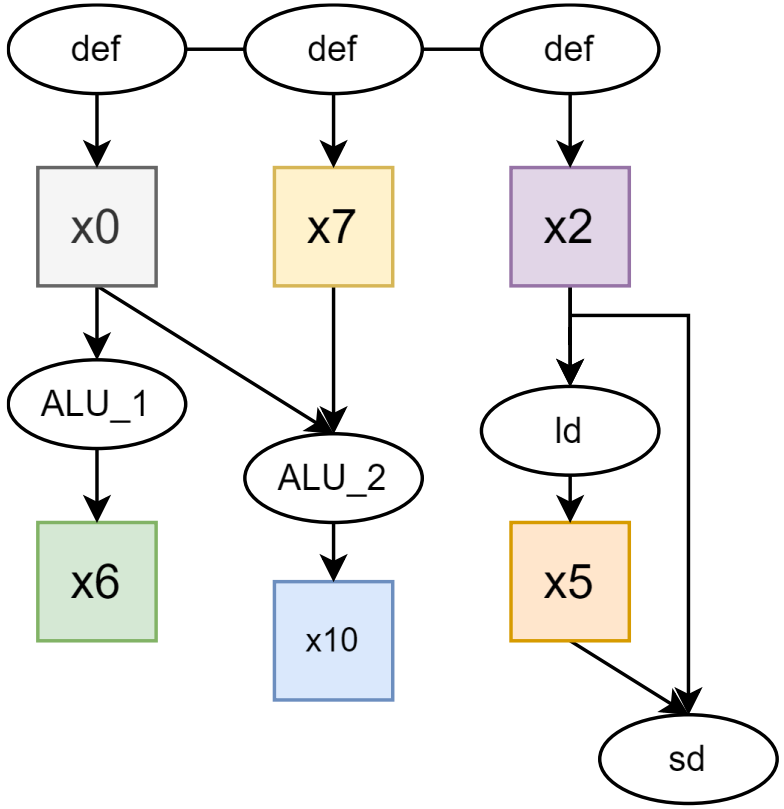
```
addi x6, x0, 42
ld   x5, 0x40(x2)
sd   x5, 0x48(x2)
add x10, x7, x0
```

Недостаточная детализация функциональных симуляторов

Рассмотрим внимательнее кусочек кода на ассемблере:

```
addi x6, x0, 42
ld   x5, 0x40(x2)
sd   x5, 0x48(x2)
add x10, x7, x0
```

Обратите внимание на зависимость операций от значений регистров

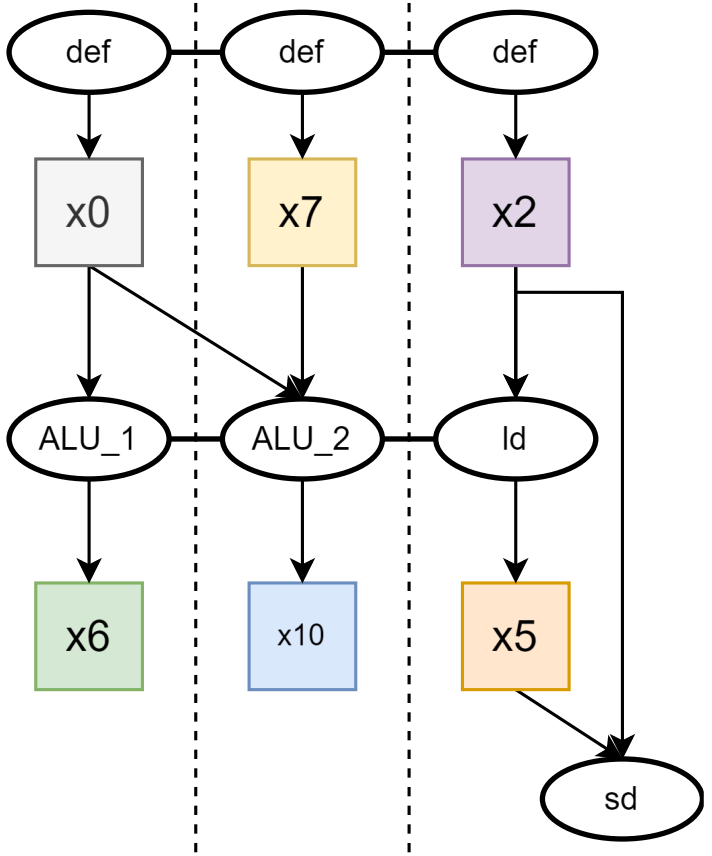


Недостаточная детализация функциональных симуляторов

Рассмотрим внимательнее кусочек кода на ассемблере:

```
addi x6, x0, 42
ld   x5, 0x40(x2)
sd   x5, 0x48(x2)
add x10, x7, x0
```

Можно заметить, что эти четыре операции представляют собой работу с тремя независимыми потоками данных



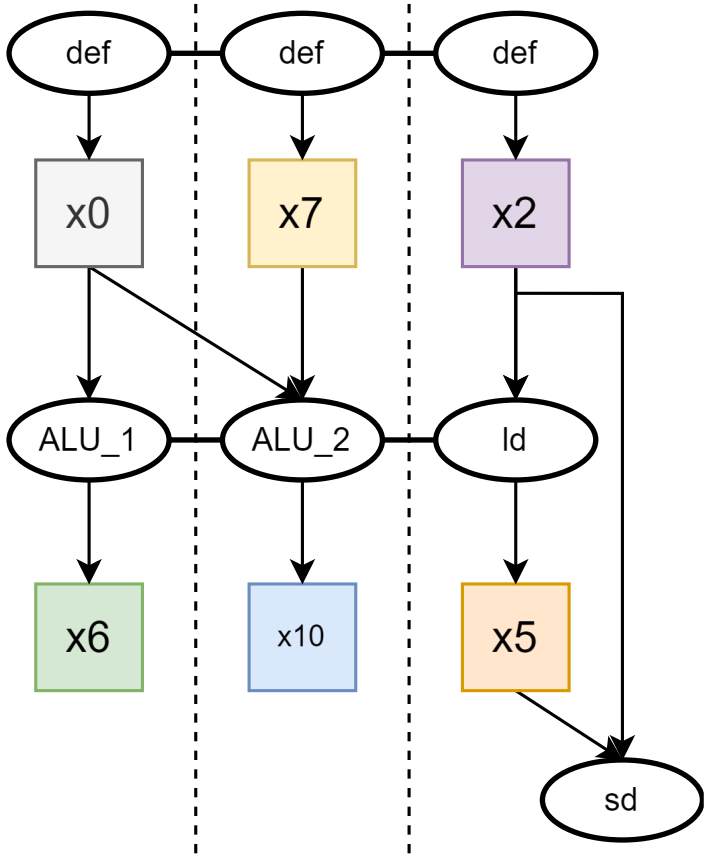
Недостаточная детализация функциональных симуляторов

Рассмотрим внимательнее кусочек кода на ассемблере:

```
addi x6, x0, 42
ld   x5, 0x40(x2)
sd   x5, 0x48(x2)
add x10, x7, x0
```

На CPU с OOO они легко могут быть переупорядочены:

```
-> x5 = mem(x2 + 0x40)
-> x6 = 42
-> x10 = x7
-> mem(x2 + 0x48) = x5
```



Недостаточная детализация функциональных симуляторов

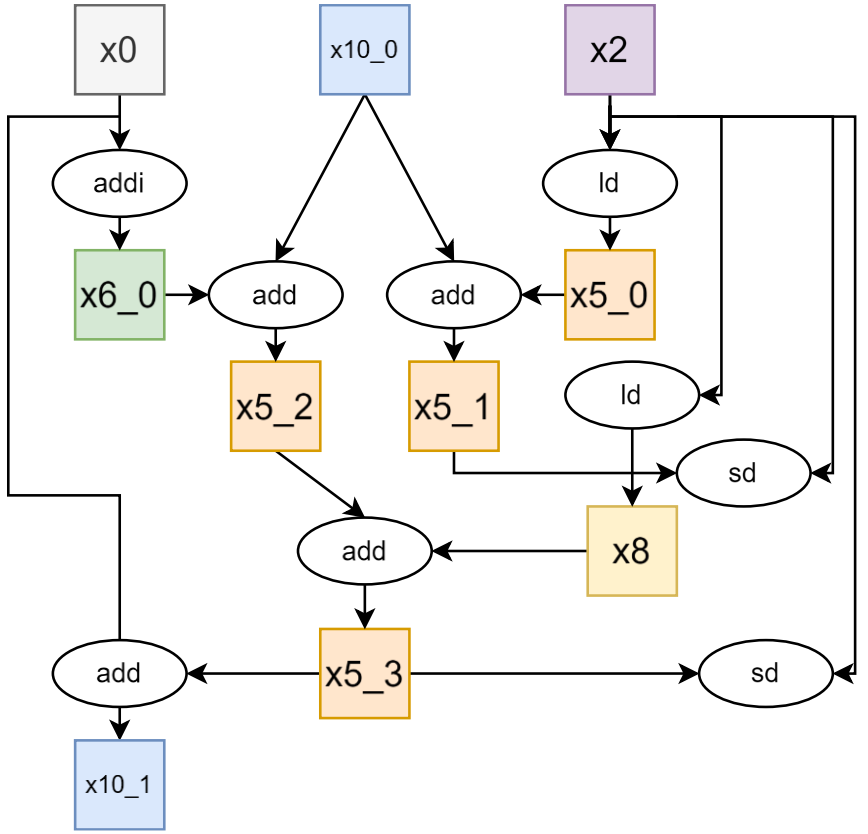
Рассмотрим кусочек кода на ассемблере:

```
addi x6, x0, 42
ld   x5, 0x40(x2)
add  x5, x10, x5
sd   x5, 0x40(x2)
add  x5, x6, x10
ld   x8, 0x48(x2)
add  x5, x8, x5
sd   x5, 0x56(x2)
add x10, x5, x0
```

Недостаточная детализация функциональных симуляторов

Расхождения могут быть существеннее более сложных примерах:

```
addi x6, x0, 42
ld   x5, 0x40(x2)
add  x5, x10, x5
sd   x5, 0x40(x2)
add  x5, x6, x10
ld   x8, 0x48(x2)
add  x5, x8, x5
sd   x5, 0x56(x2)
add x10, x5, x0
```



Чиним недостатки

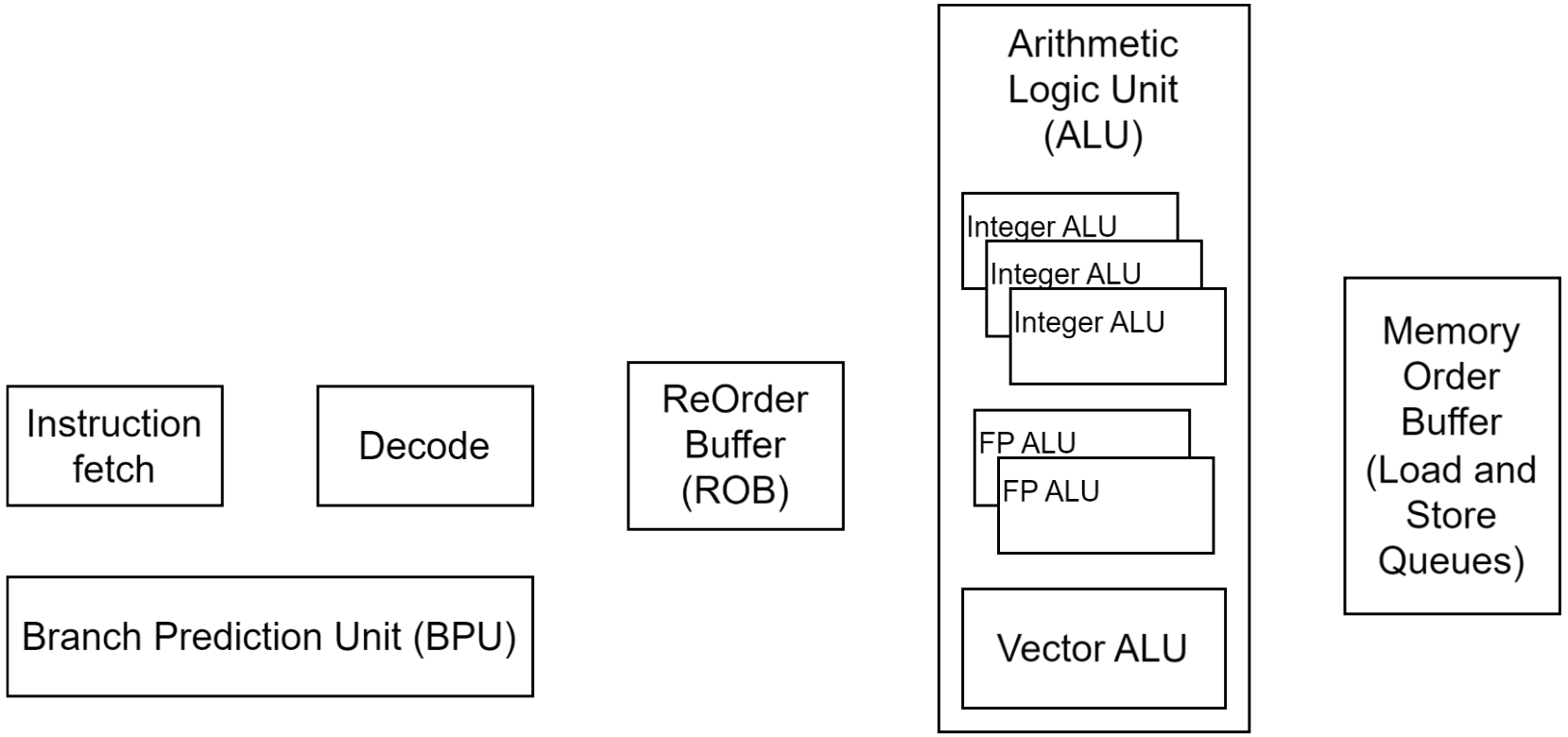
- Для исследования производительности нашей микроархитектуры точности функциональных симуляторов может быть недостаточно
  - Точность BPU
  - Качество OOO
  - Точность prefetcher’a
  - Эффективность multi-issue конвейера
- Если главным недостатком является низкая точность, то давайте эту точность повысим

Микроархитектурный симулятор

- Если главным недостатком функционального симулятора является низкая точность, то давайте эту точность повысим
- Процессор состоит из блоков, состояние которых мы хотим знать – давайте эмулировать такие блоки
- В первом приближении детали реализации таких блоков не важны
- Важно поведение и интерфейс между блоками

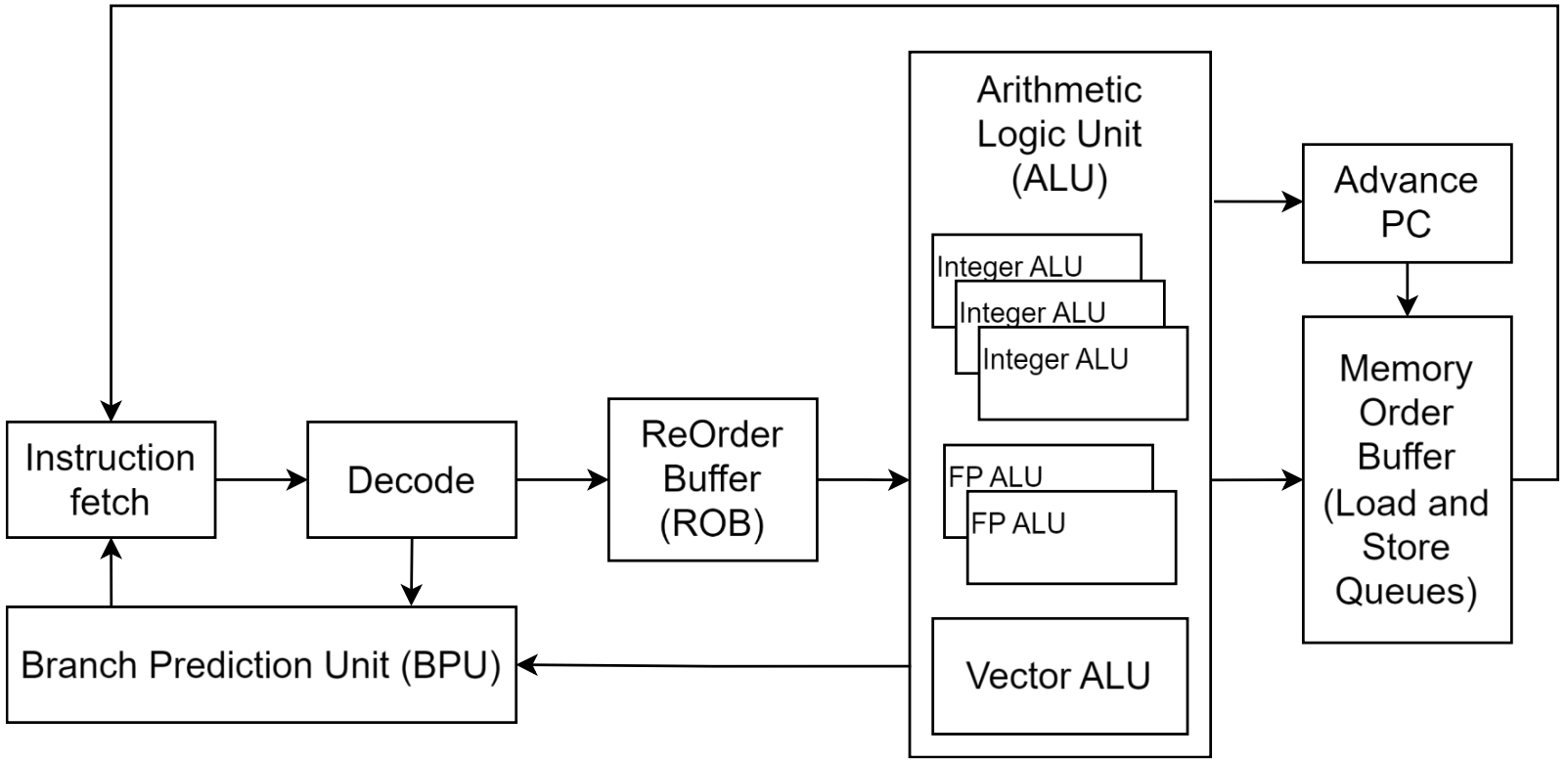
Микроархитектурный симулятор

Например, микроархитектура современного суперскалярного процессора состоит примерно из следующих блоков:



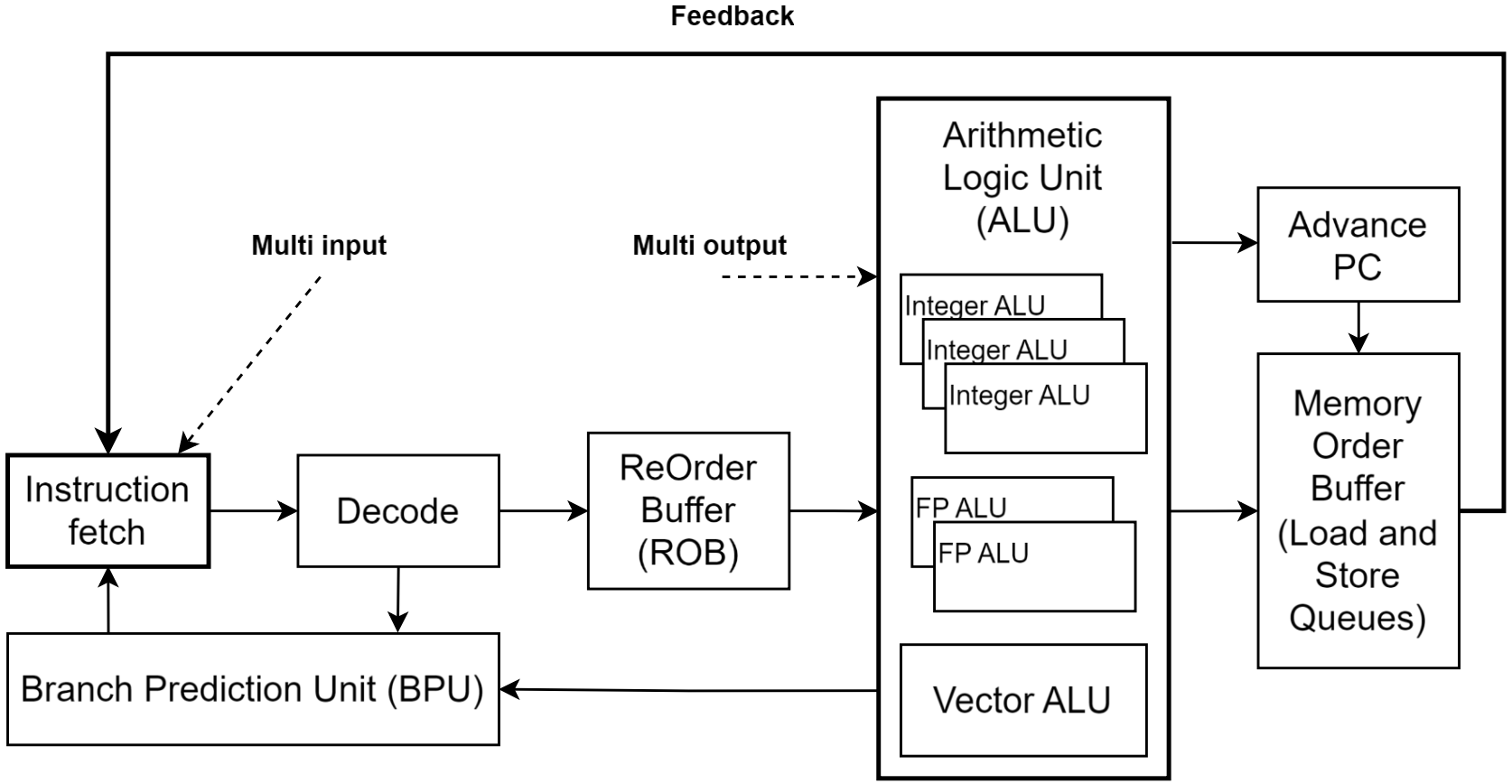
Микроархитектурный симулятор

Эти блоки могут быть причудливо связаны:



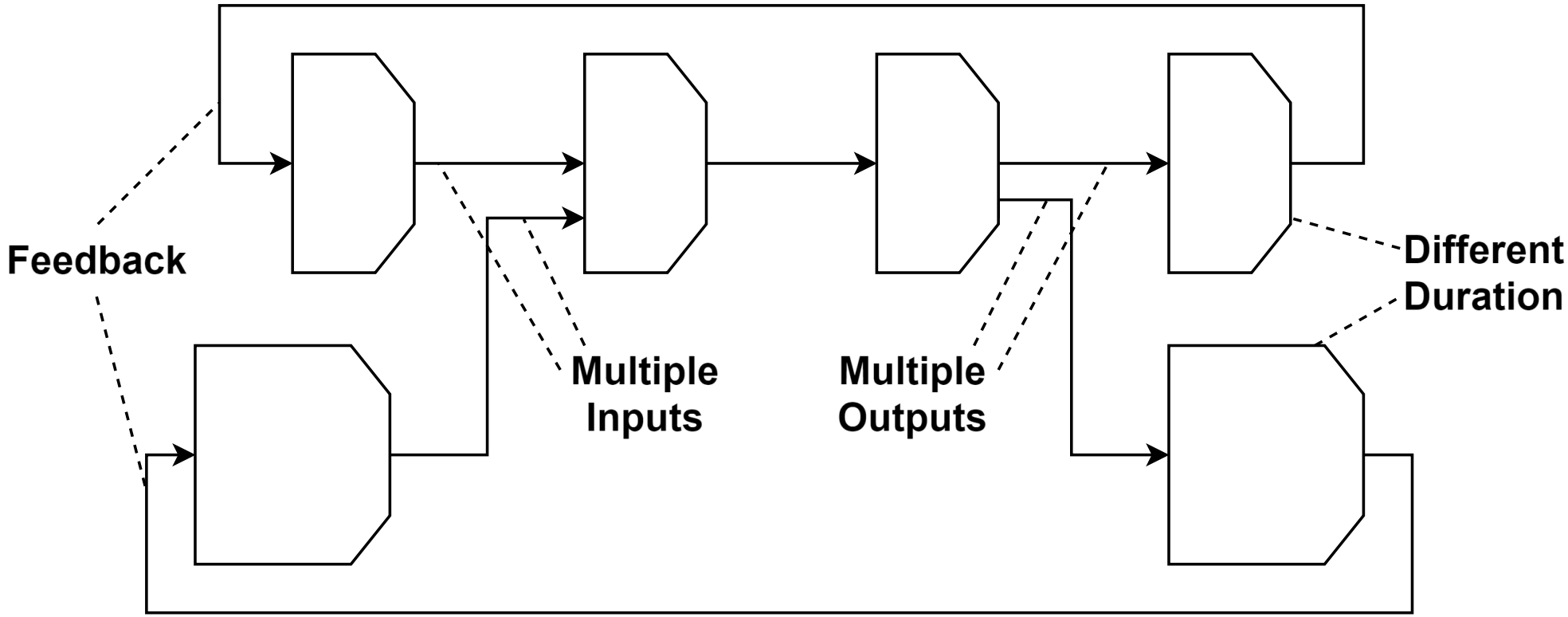
Микроархитектурный симулятор

Эти блоки могут быть причудливо связаны:



Микроархитектурный симулятор

Обобщая подобную схему:



Детальность реализации микроархитектурных блоков

- По детальности реализации микроархитектурных блоков такие модели можно классифицировать на:
  - Транзакционные
  - Cycle-accurate
- Как правило самый сложный параметр при повышении детализации – выдерживание таймингов, соответствующих дизайну

Что значит транзакция?

В контексте моделирования ЭВМ:

Транзакция – это вызов одним компонентом операции над другим компонентом

- Примером транзакции является очистка конвейера из-за неверно предсказанного перехода:
  - Как только условие перехода посчитано, оно сравнивается с предсказанием
  - Если предсказание неверно нужно очистить конвейер от загруженных по ошибке инструкций
  - Специальный блок посылает транзакции, которые запускают сброс стадий конвейера

Transaction-Level Model (TLM)

- Транзакционные модели так же можно классифицировать:
  - Untimed
    - Тайминги никак не учитываются
    - Все транзакции моделируются как атомарные события
    - Используется для моделирования функциональных свойств транзакционных протоколов
  - Loosely timed
    - Каждой транзакции соответствует время ее выполнения, благодаря чему можно грубо оценить временные характеристики
    - Количество транзакций совпадает с аппаратурой
    - Очередность транзакций может отличаться от аппаратуры
  - Approximately timed
    - Количество и очередность транзакций совпадает с аппаратурой
    - Можно точно измерить погрешность модели

Обсуждение: реализуем модель с учетом таймингов

Проблемы:

- Схожие операции могут требовать разного времени для разных блоков
- Как проверить готовность «медленных» блоков?
- Новый результат появляется с новым тактом

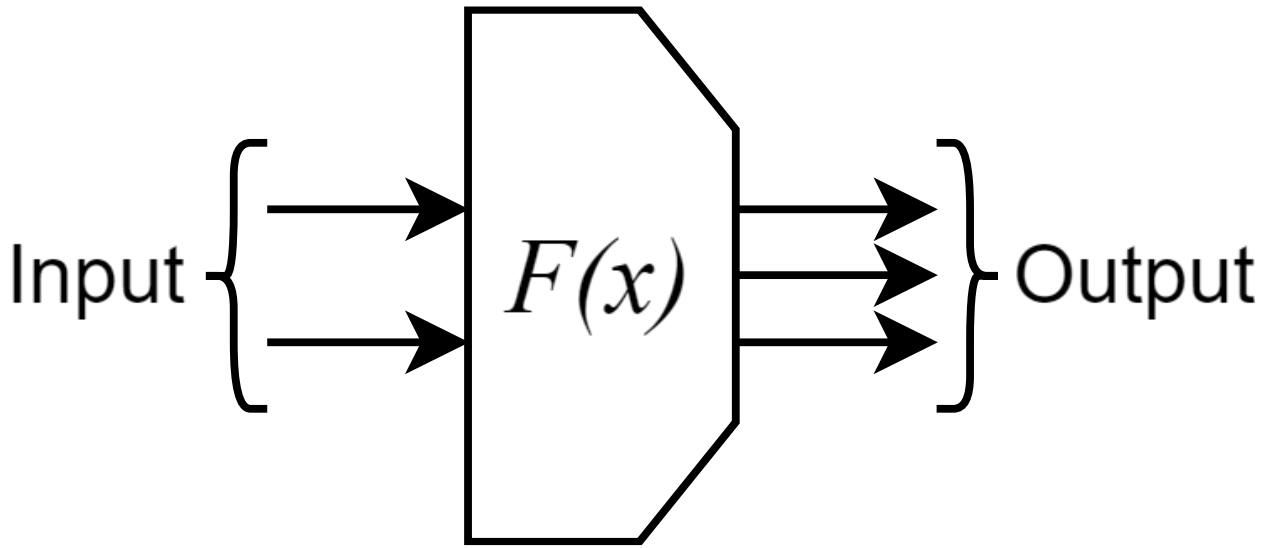
Реализуем модель с учетом таймингов

Решение: разделение обязанностей:

- Функции блоков
- Время, необходимое для вычисления функции блока
- Внутреннее состояние блока

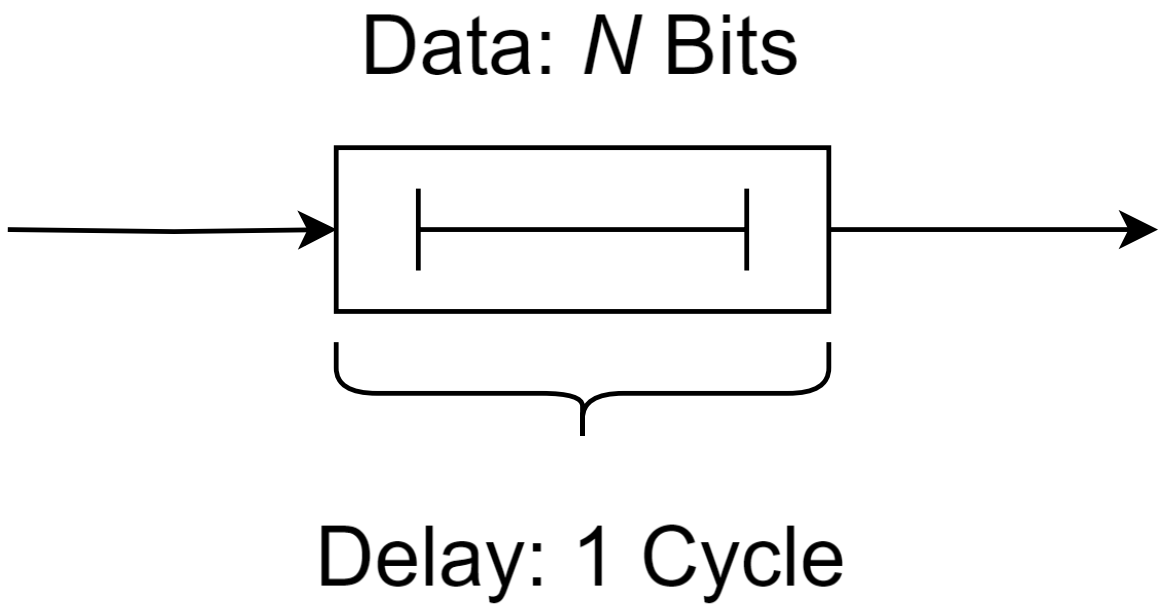
Функция блока

Мгновенно вычисляет результат, если входные данные готовы



Порт

Порт это очередь с фиксированной задержкой

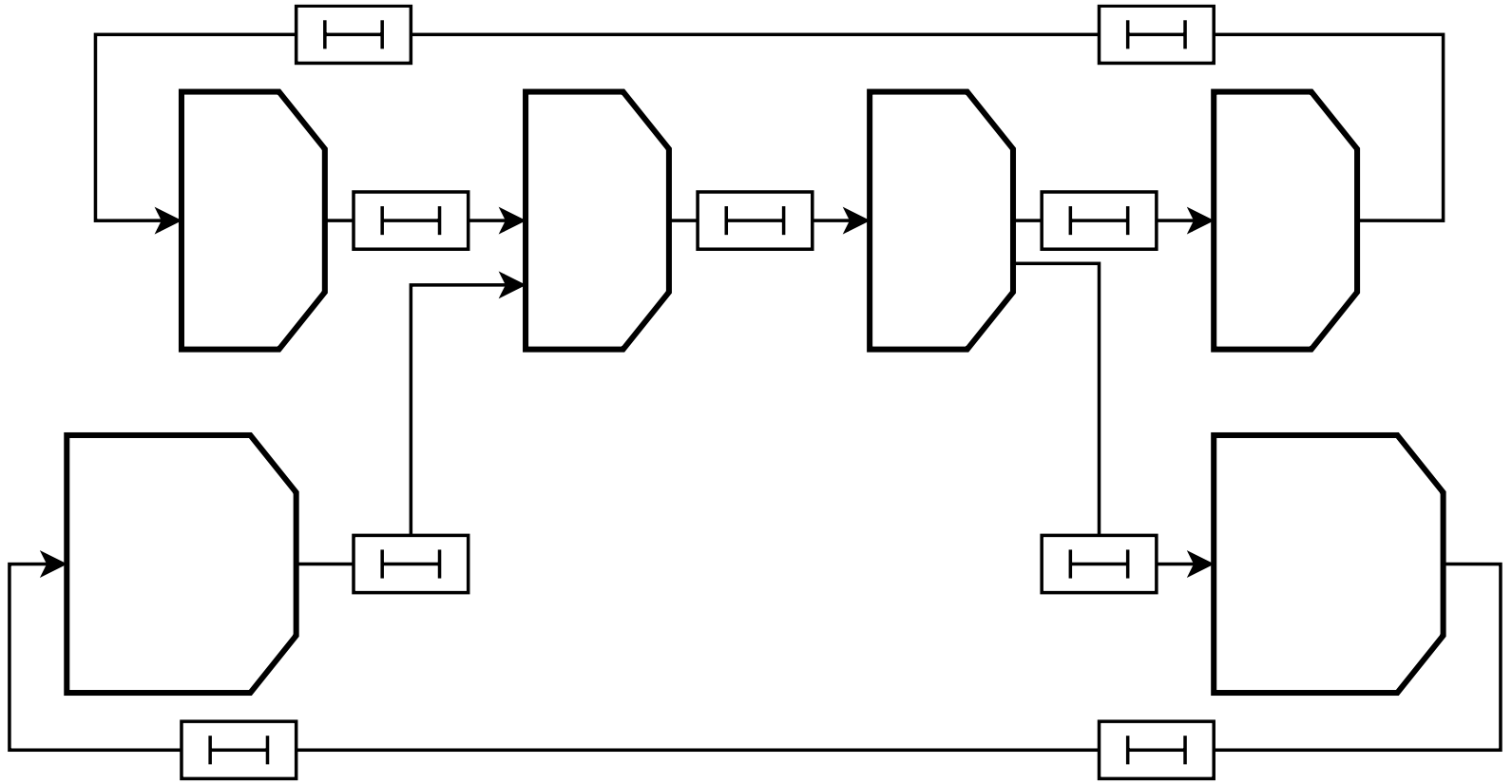


Правила соединений

- Функции не могут быть соединены напрямую
- Фазы симуляции чередуются:
  - Симуляция функций
  - Симуляция передачи результатов

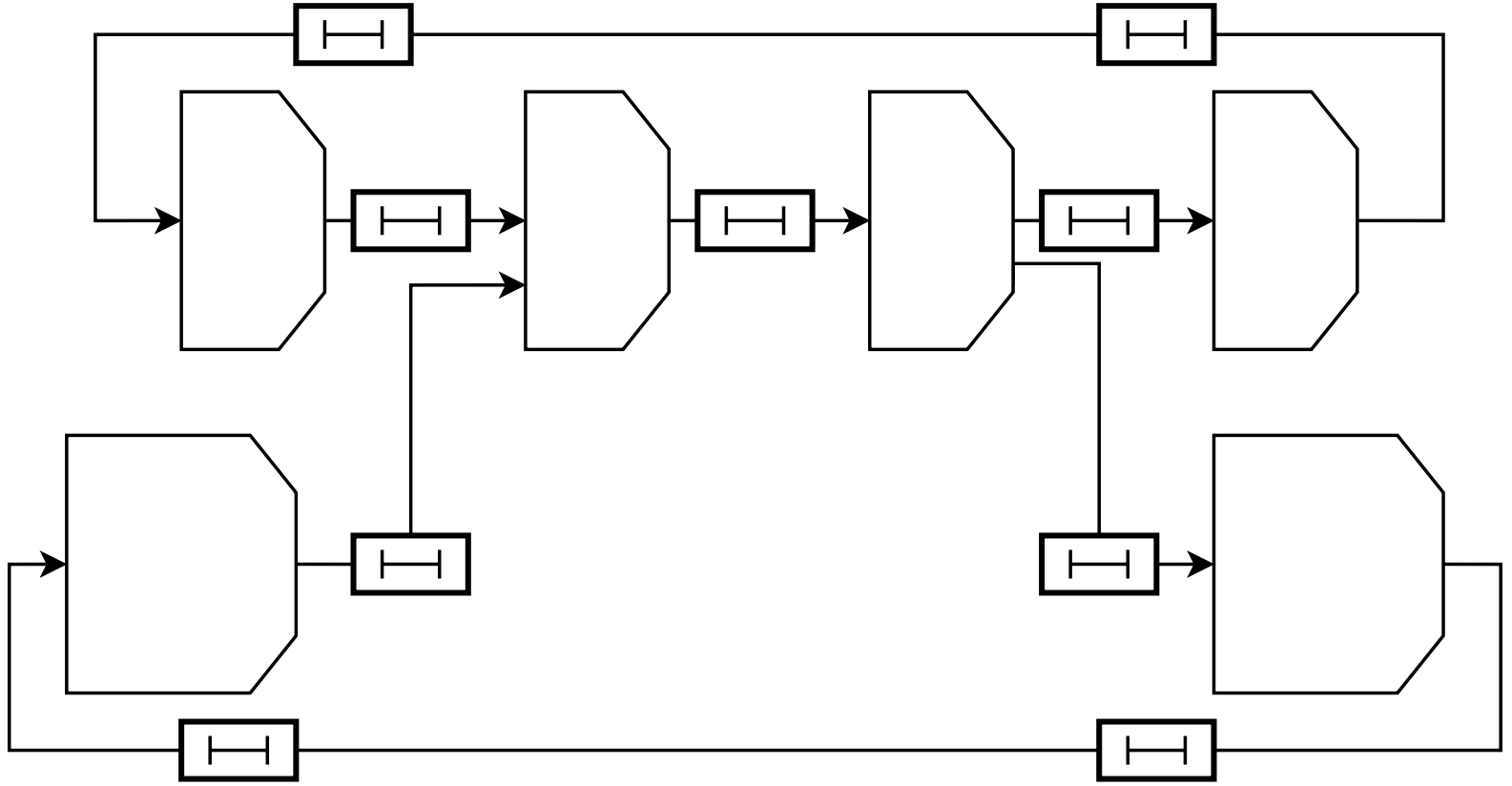
Transaction-Level Model +Time

Первая фаза симуляции



Transaction-Level Model +Time

Вторая фаза симуляции



Микроархитектурные симуляторы RISC-V

- Существует большой пласт образовательных проектов
- Самым крупным OS решением является GEM5
- Так как у каждого вендора микроархитектура отличается, то GEM5 это скорее не симулятор, а фреймворк
- Просто скачать и запустить GEM5 для своего процессора не получится, потому что GEM5 надо настроить/пропатчить, чтобы он моделировал свойства желательной системы
- Запускаем GEM5:

```
time gem5 /opt/gem5/configs/riscv-coremark.py
```

Теперь довольны?

- С помощью разных TLM можно исследовать различные свойства микроархитектуры
- Но это всего лишь аппроксимирующие модели
- Что если мы хотим посмотреть как с точностью до такта ведет себя процессор при прогоне бенчмарка?
- Что если мы хотим получить более точную оценку потребляемой мощности?

Выход – потактовая модель

- Потактовая модель (cycle-accurate) – точная модель RTL уровня
- Есть два основных подхода по созданию потактовых моделей
  - Сгенерировать из RTL описания
  - Написать на C/C++
- Какие недостатки вы видите у каждого из подходов?

Критикуем и не предлагаем

RTL модель

- Преимущества
  - На 100% соответствует RTL описанию
  - Не требует дополнительных затрат на разработку – есть RTL, есть модель
- Недостатки
  - RTL как правило готов только к самому выпуску процессора и заранее использовать такую модель невозможно
  - Все RTL симуляторы или медленные или ОЧЕНЬ медленные

C/C++ модель

- Преимущества
  - Быстрее RTL модели
- Недостатки
  - Необходимо поддерживать в актуальном состоянии с RTL
  - Высокая стоимость разработки, могут позволить только крупнейшие компании

Суровая реальность

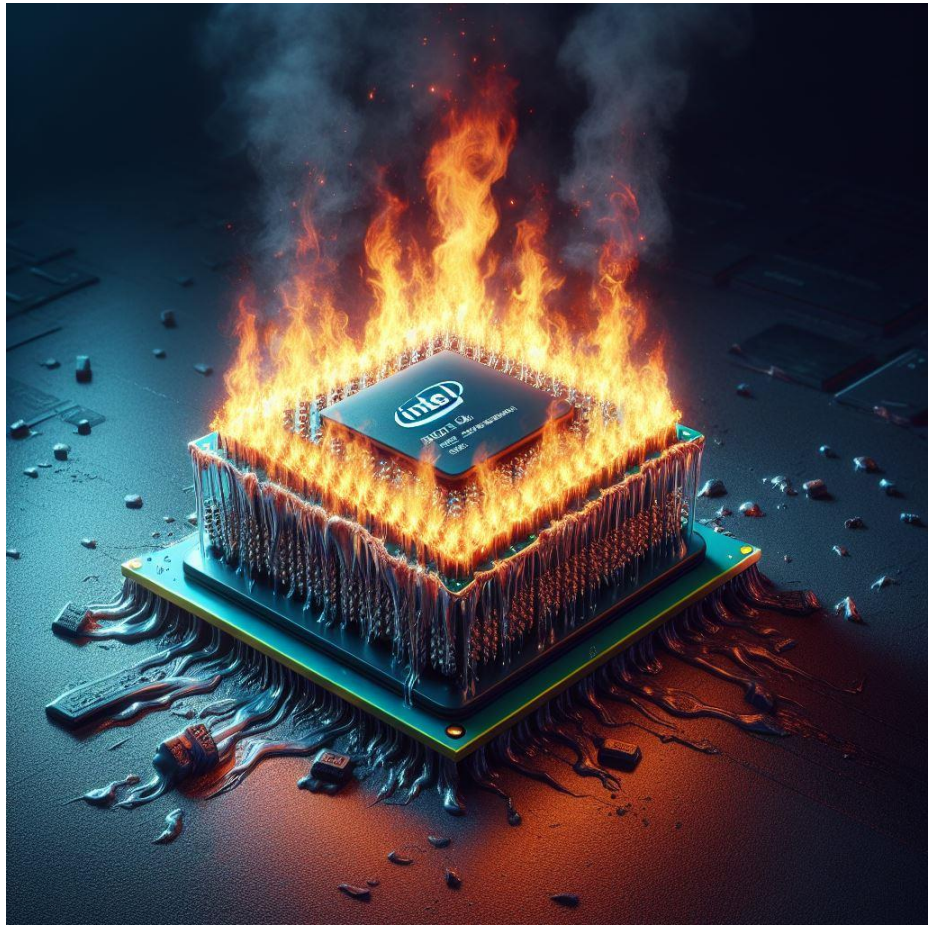
- Наличия хорошей cycle-approximate и функциональной модели достаточно для большинства запросов ПО на ранних этапах разработки, когда RTL еще не готов
- RTL симулятор используется только для
  - Верификации небольшими тестами
  - Исследований задержек для более точной настройки cycle-approximate моделей

Ниже некуда: физический уровень

Любая (даже невероятно сложная) интегральная схема – всего лишь электрическая схема

Это еще один слой абстракции

И его тоже необходимо верифицировать на предмет ошибок



Intel troubles

Симуляторы физического уровня

Симуляторы физического уровня позволяют проводить анализ основных аналоговых величин:

- AC-анализ (анализ по переменному току)
- DC-анализ (анализ по постоянному току) для слабых сигналов
- Анализ шумов
- Анализ переходных процессов

Основные симуляторы: **SPICE**, **Spectre**, **Eldo…**

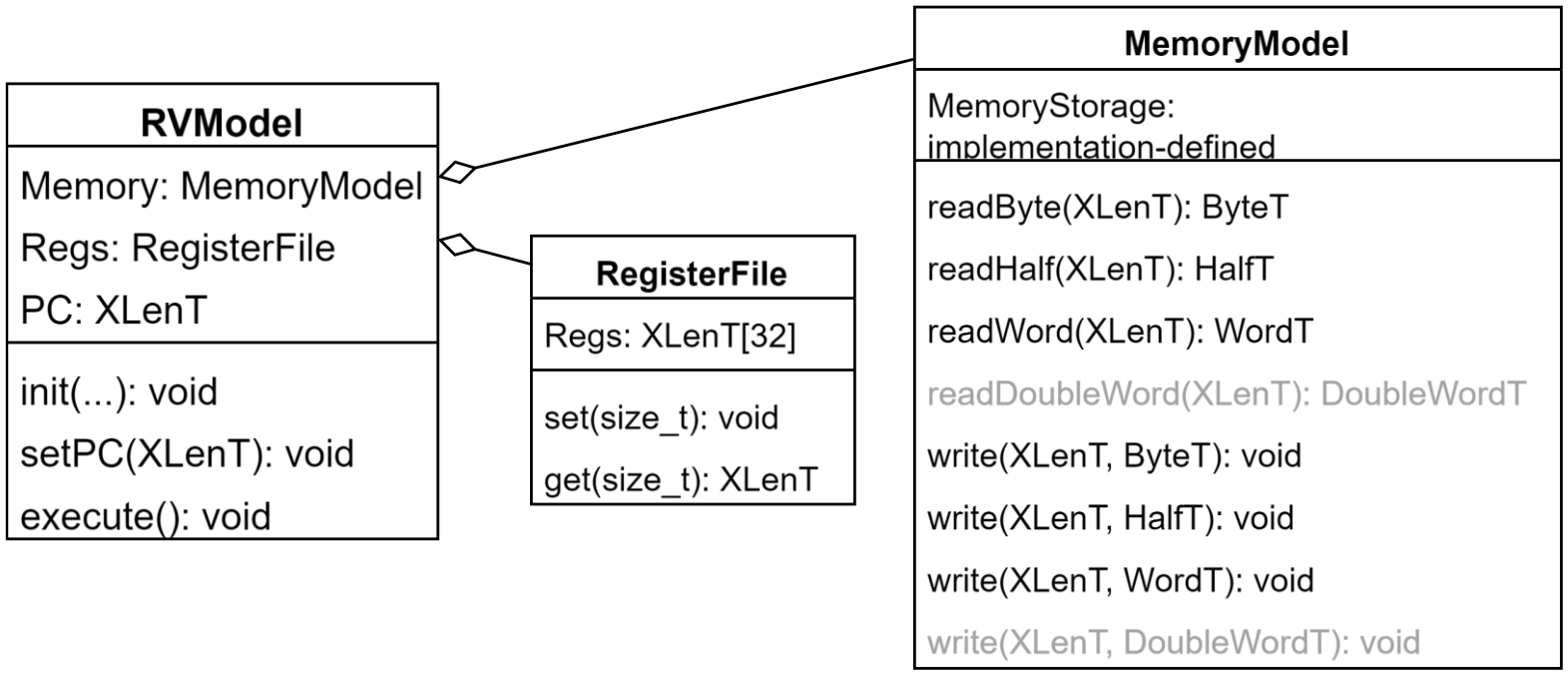
To be continued …

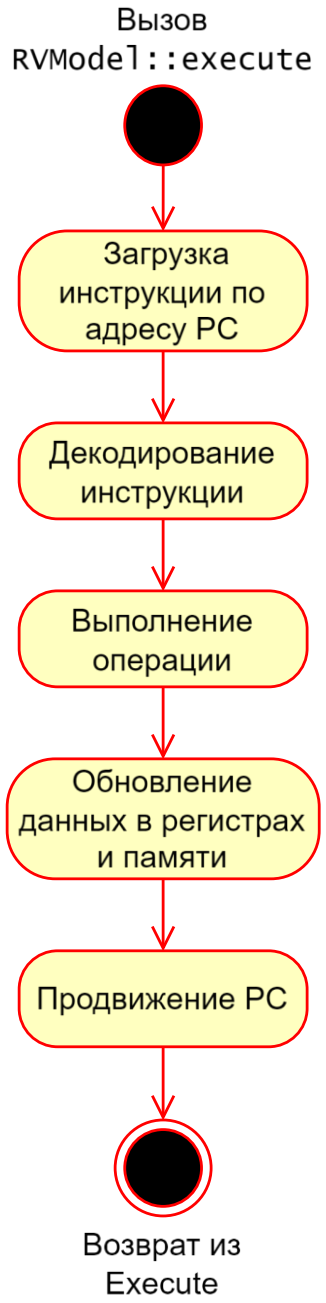
На следующем занятии узнаем

- На какую головную боль готовы люди, чтобы ускорить функциональный симулятор
- Узнаем как можно в некоторых случаях получается так, что программа на симуляторе исполняется быстрее, чем на хосте

Задание: пишем модель

Реализуйте модель rv32i процессора





Задание: пишем модель

Реализуйте модель rv32i процессора, кроме инструкций `ecall`, `ebreak` и `fence.i`

По возможности попробуйте предусмотреть:

- Добавление расширений и возможность их включения/выключения
- Конфигурацию 32/64 bit

Список литературы

- The RISC-V Instruction Set Manual Volume I Unprivileged Architecture Version 20240411
- Hennessy J. L., Patterson D. A. Computer architecture: a quantitative approach. – Morgan kaufmann, 2017.
- Основы программного моделирования ЭВМ: Учебное пособие / Речистов Г.С., Юлюгин Е.А., Иванов А.А., Шишпор П.Л., Щелкунов Н.Н., Гаврилов Д.А. — 2-е изд., испр. и доп. — Издательство МФТИ, 2013.