Средства симуляции ЦП и ОС и изучение поведения программ

Лекция №6

Державин Андрей
Шурыгин Антон

- **Квиз**
  - Режим приложения
  - Симуляция полной платформы

Код викторины:

00709081

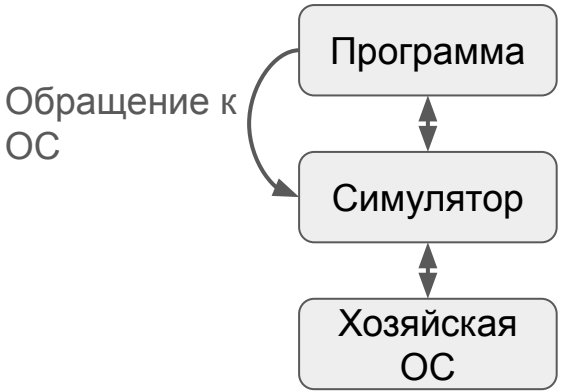
- Квиз
- **Режим приложения**
- Симуляция полной платформы

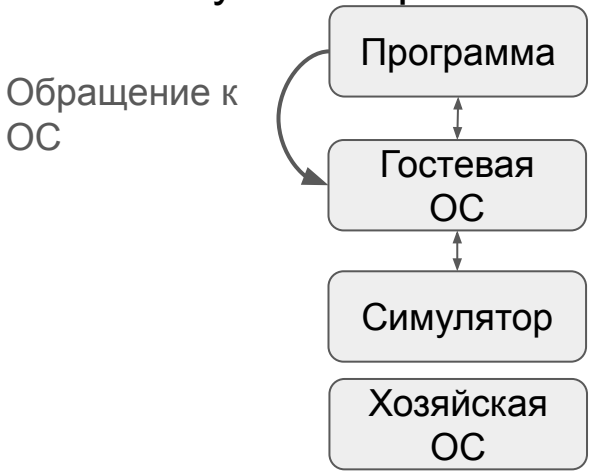
Режим приложения

- Написали интерпретатор (и бинарный транслятор)
- Что он умеет?
  - Выполянть арифметические операции над регистрами
  - Совершать условые/безусловные переходы
  - Работать с памятью
  - Выходить (завершать работу по ebreak)
- Как часто пишутся подобного рода программы?
  - Обычно программы взаимодействуют с операционной системой
- Симулятор должен в каком-то виде эмулировать поведение ОС

Режим приложения

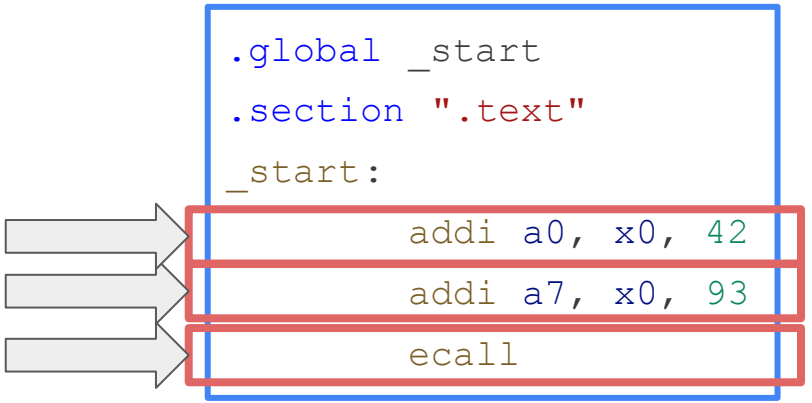
- Каким образом приложение взаимодействует с ОС?
  - Системные вызовы - открытие файлов, чтение/запись, ввод-вывод в потоки
- 2 способа эмуляции ОС:
  - Эмуляция системных вызов (частично с помощью средств хозяйской ОС)
  - Эмуляция полной платформы с дальнейшим запуском ОС - full-system simulation
- В симуляторах режима приложения обычно используется первый метод





Режим приложения. Системные вызовы

- Как совершается системный вызов?
  - Вспомним первое ДЗ (и man syscall)
- ecall - команда системного вызова
- a7 содержит номер системного вызова
  - таблицы в открытом доступе
- a0-a6 содержат аргументы вызова
- Какой вызов на экране?
  - exit(42)



Режим приложения. Системные вызовы

- Какие системные вызовы могут понадобиться?
  - завершение программы (exit)
  - работа с файлами (открытие, закрытие, чтение, запись, fstat…)
  - ввод-вывод через стандартные потоки (stdin, stdout, stderr)
  - работа с памятью (mmap, …)
  - работа с временем
  - работа с потоками (иногда)

Режим приложения. Системные вызовы

- Давайте передавать системные вызовы вместе с аргументами на хозяйскую систему!
  - Например через функцию long syscall(long number, ...);
- У нас все хорошо?
  - Нужно транслировать номера системных вызовов (в общем случае они различаются)
  - В качестве аргументов могут передаваться гостевые адреса, их нужно транслировать
  - Не все системные вызовы стоит действительно исполнять
    - Как минимум exit()
- Нужно писать обработчики для каждого системного вызова

Режим приложения. Системные вызовы

- Обсудим некоторые “популярные” системные вызовы
- exit
  - Должен завершать работу симулятора (возможно установить статус возврата)
- open
  - Можно оставить как есть
  - Для воспроизводимости реализуется детерменированный алгоритм назанчения гостевых fd
- write, read
  - Нужно транслировать аргументы с адресами
- mmap
  - Выделить гостевые адреса

Режим приложения. Стек

- Для чего используется стек?
  - Локальные переменные
  - Адрес возврата
- Скомпилируем пример:

```
int foo(int a) {
  int b = a + 1;
  return b;
}
```

riscv-gcc

sp == ?

```
<foo>:
  addi    sp,sp,-48
  sd      s0,40(sp)
  addi    s0,sp,48
  addi    a5,a0,0
  sw      a5,-36(s0)
  lw      a5,-36(s0)
  addiw   a5,a5,1
  sw      a5,-20(s0)
  lw      a5,-20(s0)
  addi    a0,a5,0
  ld      s0,40(sp)
  addi    sp,sp,48
  jalr    zero,0(ra)
```

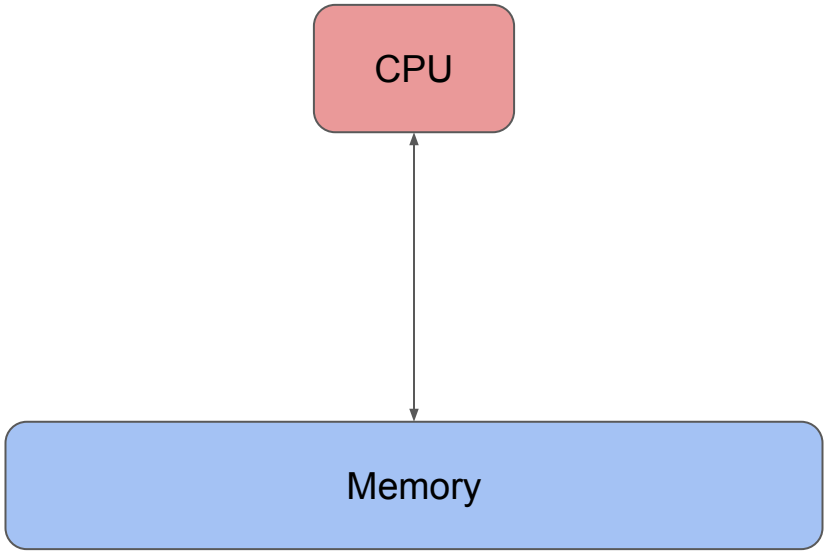
Режим приложения. Стек

- Нужно задавать начальное значение sp

- Квиз
- Режим приложения
- ➢ **Симуляция полной платформы**

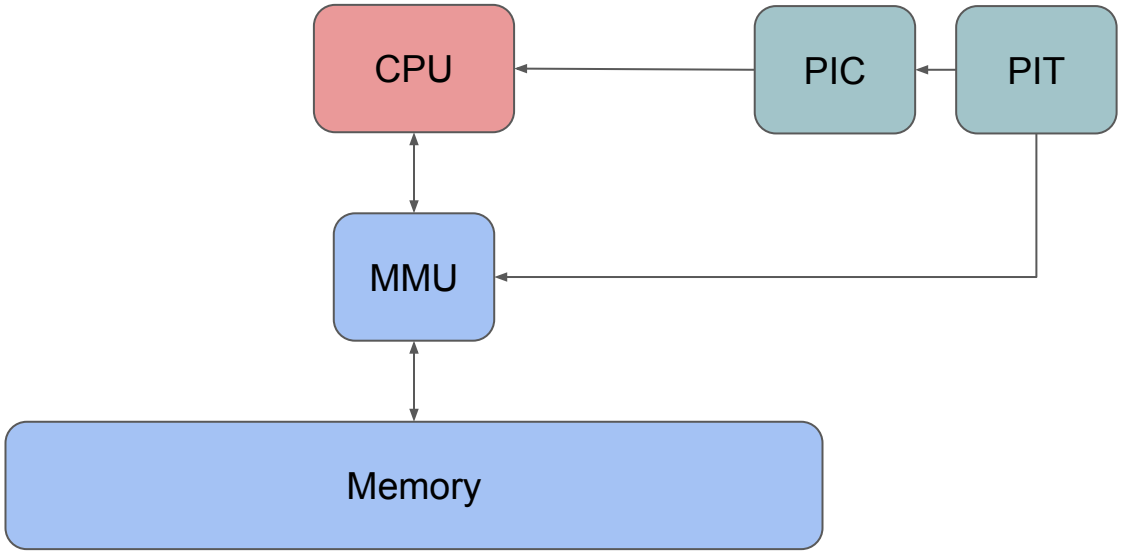
Симуляция полной платформы

- Какие компоненты есть в нашей модели?
- Какие есть компоненты у реальной системы?



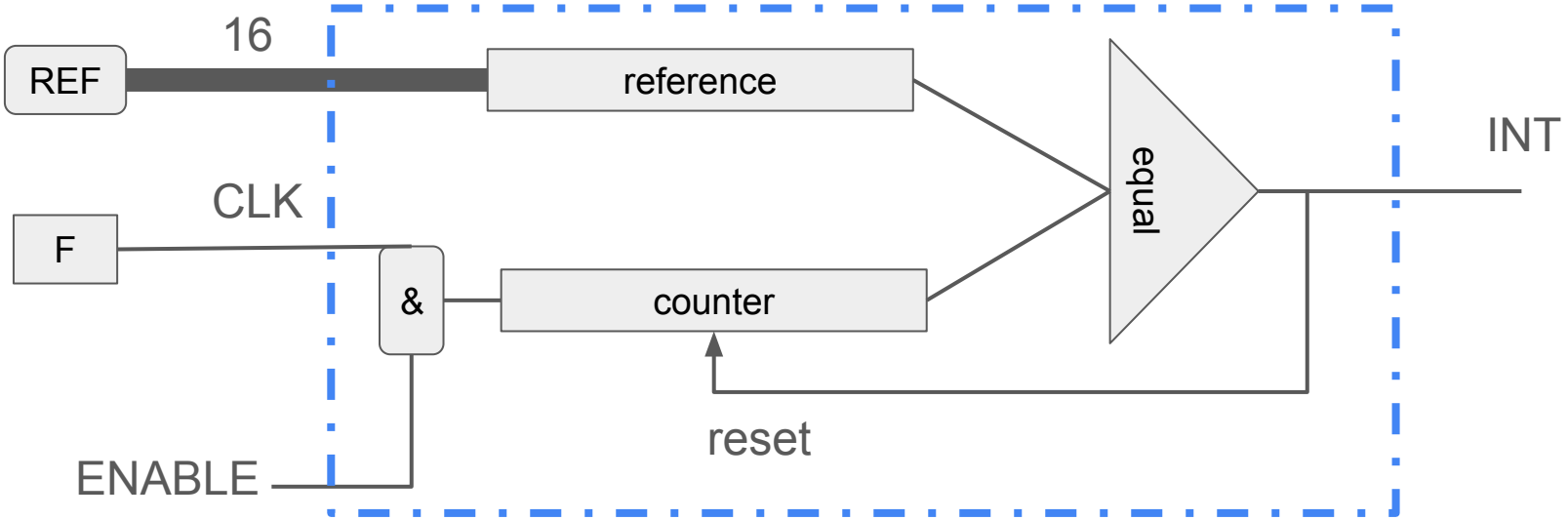
Симуляция полной платформы

- PIC - контроллер прерываний, PIT - таймер



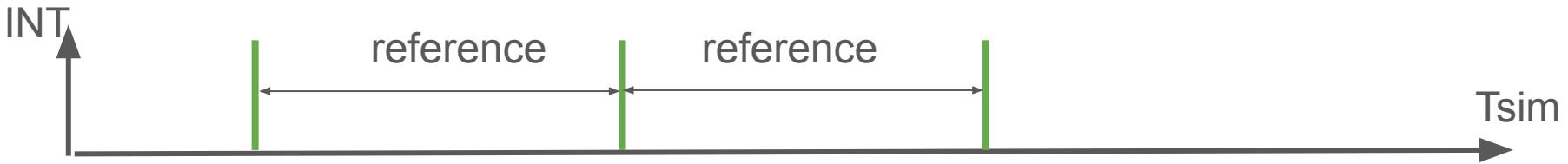
Симуляция полной платформы. Таймер

- Часто встречающееся устройство
- Задача - сгенерировать сигнал по истечении времени



Симуляция полной платформы. Таймер

- Как можно реализовать модель таймера?
- Реализуем метод on_clk()
- Симулятор на каждом такте вызывает этот метод
- Какие могут быть проблемы?
  - Устройств может быть очень много
  - Событие INT возникает очень редко (1 раз в 100’000 тактов симуляции)
- Как можно улучшить модель?
- Рассмотрим внешние эффекты таймера



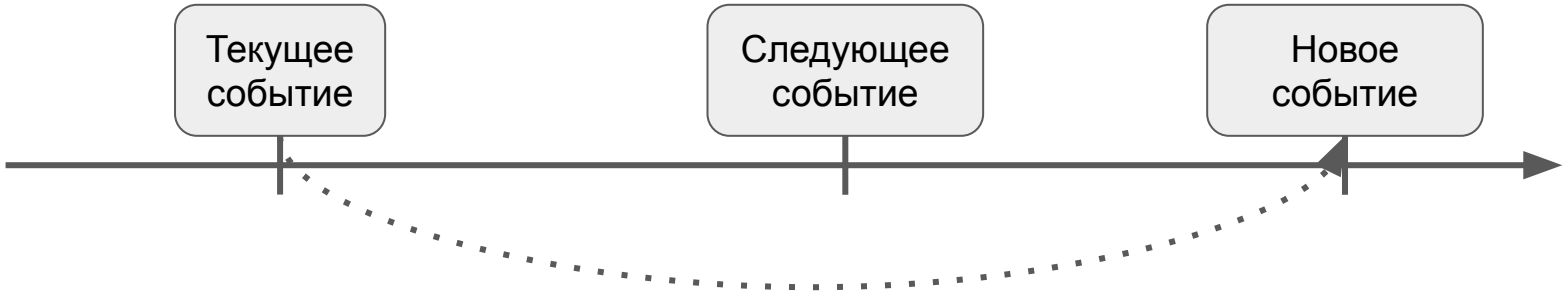
Симуляция полной платформы. Таймер

- Будем моделировать только внешние эффекты
- Храним *очередь событий*
  - Упорядочена по меткам времени
- Сигнал CLK не моделируется
- ENABLE -> 0 отменяет обработку события
- Изменение reference приводит к обновлению событий

```
struct Event {
  uint64_t time_delta;
  Dev *device;
  void (*function)(Dev *dev);
};
```

Симуляция полной платформы. Event-driven

- Обобщим модель таймера с очередью событий
- Событие - изменение состояний устройств
- Симулируемое время двигается по событиям (скачками)



Симуляция полной платформы. Event-driven

- Обсудим операции с событиями
- Добавление
  - Добавляем событие в соответствии с приоритетом (время)
- Удаление
  - Отмена события
- Одновременные события
  - Лучше придерживаться детерминизма в порядке
- Нельзя добавлять события в прошлое

Симуляция полной платформы. Два класса моделей

- Два класса моделей
  - Execution driven
  - Event driven
- Как сочетать обе модели?