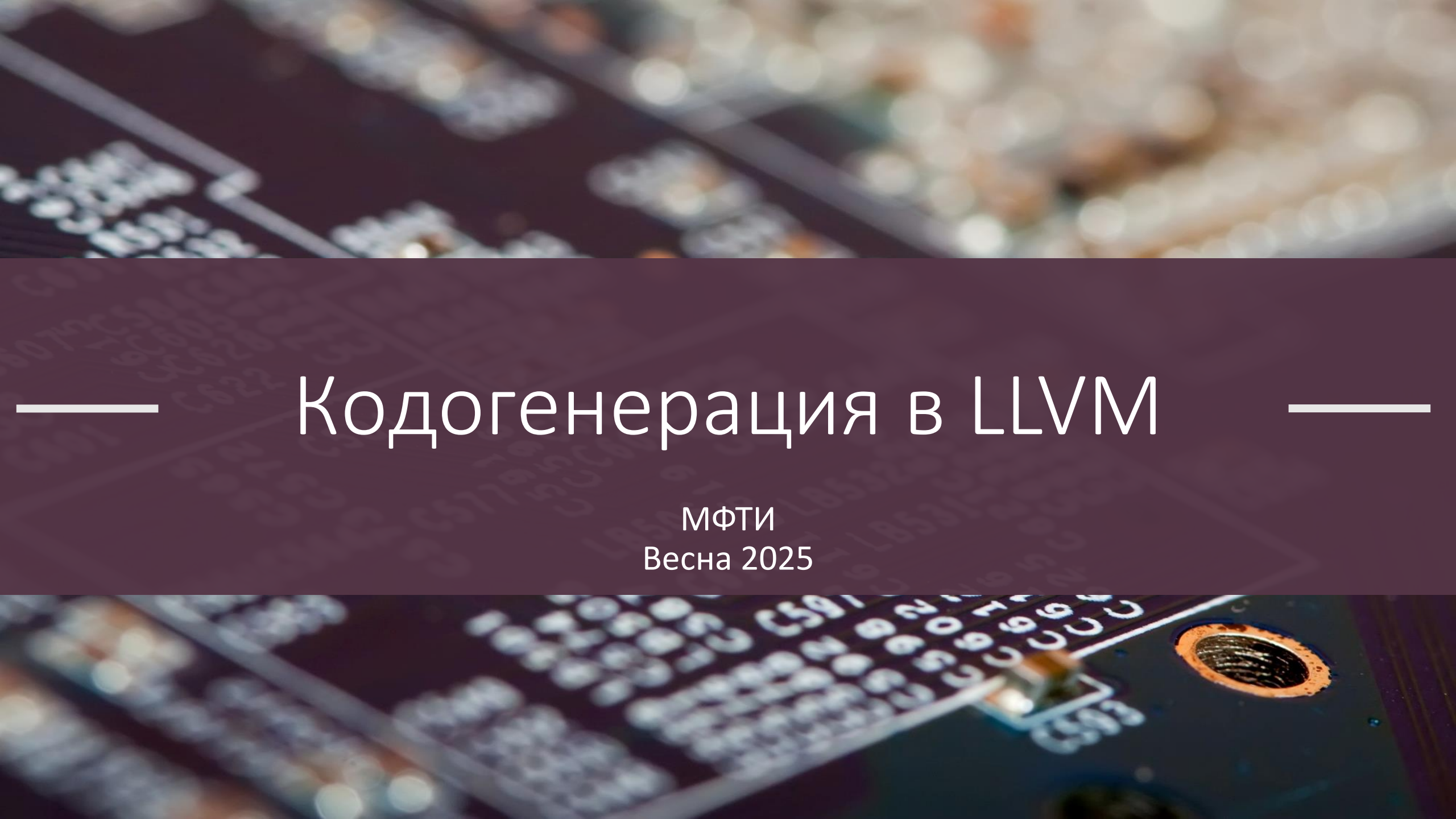
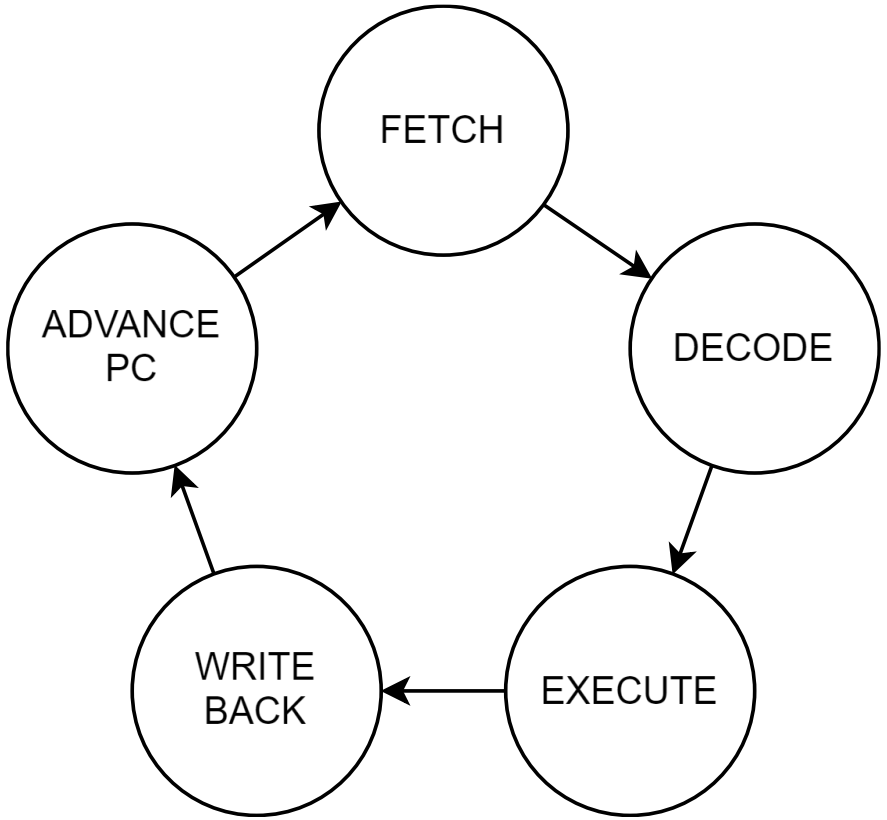


Кодогенерация в LLVM

МФТИ
Весна 2025

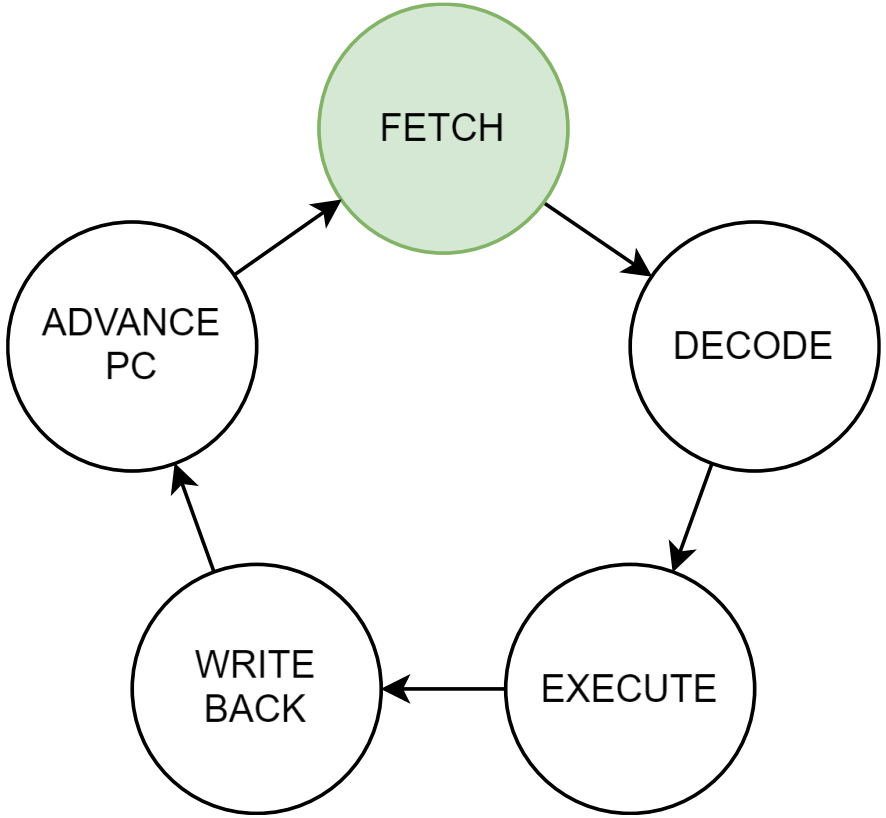
Игрушечный пример: симулятор

- Простейший функциональный симулятор состоит из нескольких фаз:
  - Fetch
  - Decode
  - Execute
  - Write back
  - Advance PC



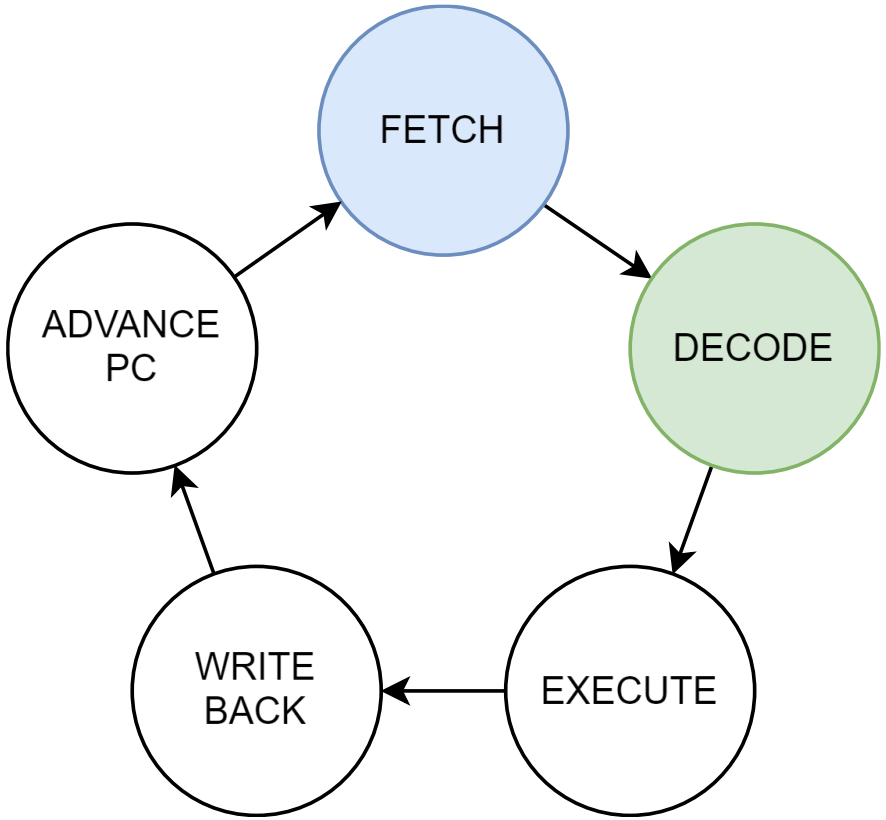
Игрушечный пример: симулятор

- На этапе **Fetch** нужно знать размер инструкции



Игрушечный пример: симулятор

- На этапе **Decode** нужно знать кодировку инструкции



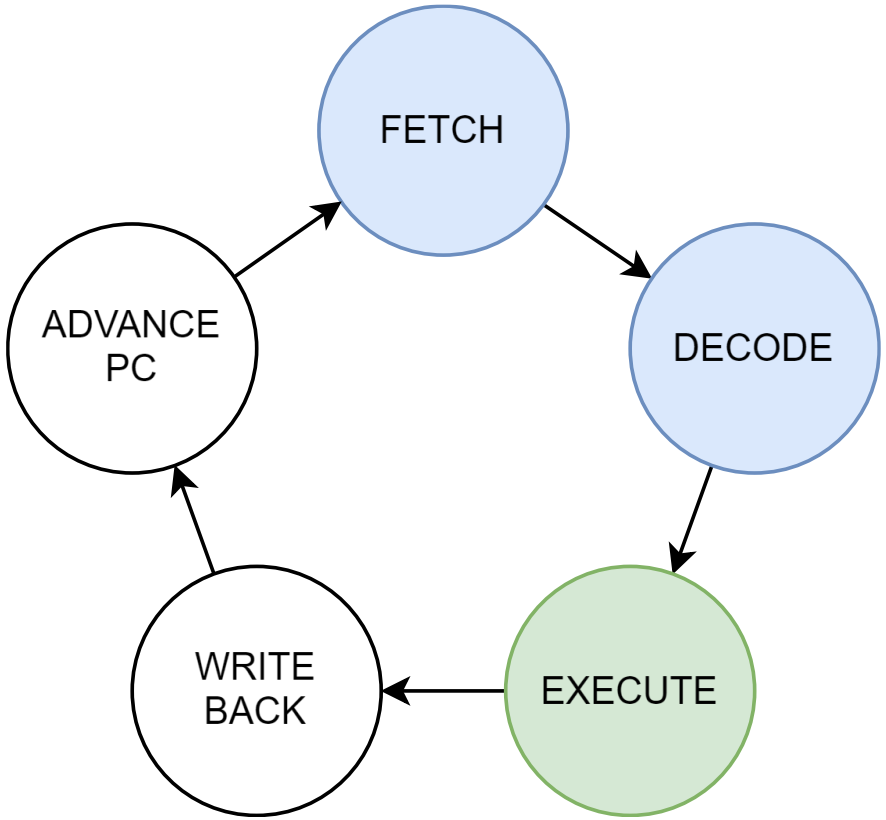
Игрушечный пример: симулятор

- На этапе **Decode** нужно знать кодировку инструкции
- Наивная реализация **Decode** использует switch-case statement
- Информация о кодировке инструкций «вшивается» в код
- Нельзя переиспользовать для кодирования инструкций

```
switch (inst & mask1) {
  case OP_CLASS_1: {
    switch (inst & mask2) {
      case OP_1:
        do_op1(inst);
        break;
…
}
```

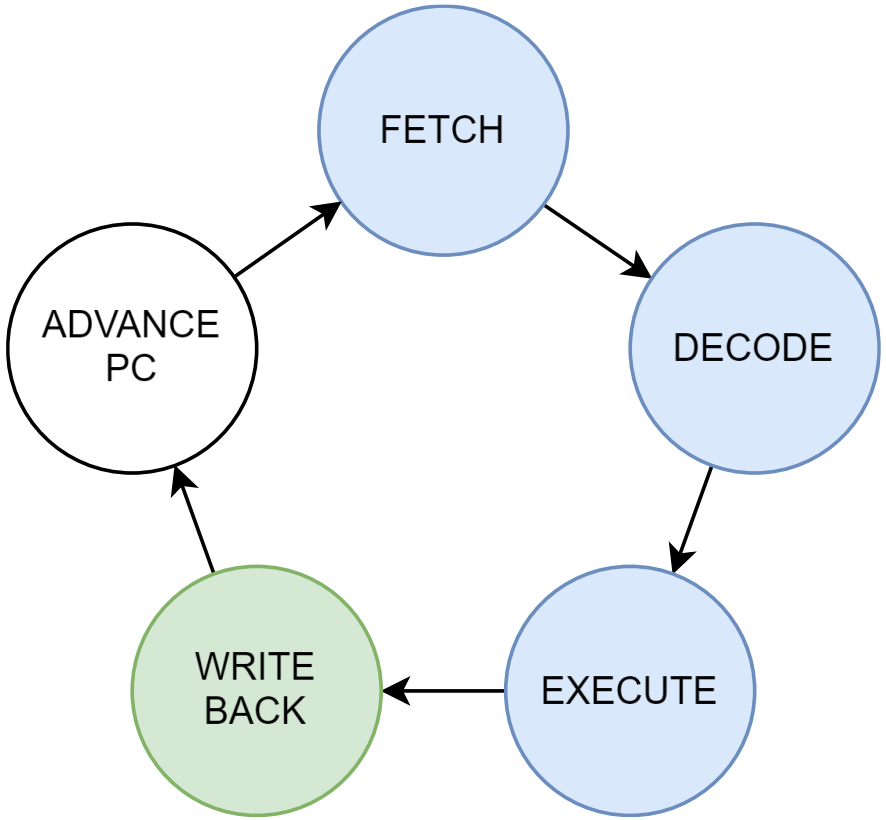
Игрушечный пример: симулятор

- На этапе **Execute** нужно знать семантику инструкции
- Этот этап больше всего отличает симулятор от кодека



Игрушечный пример: симулятор

- На этапе **Write back** нужно знать куда записывается результат выполненной инструкции



Игрушечный пример: симулятор

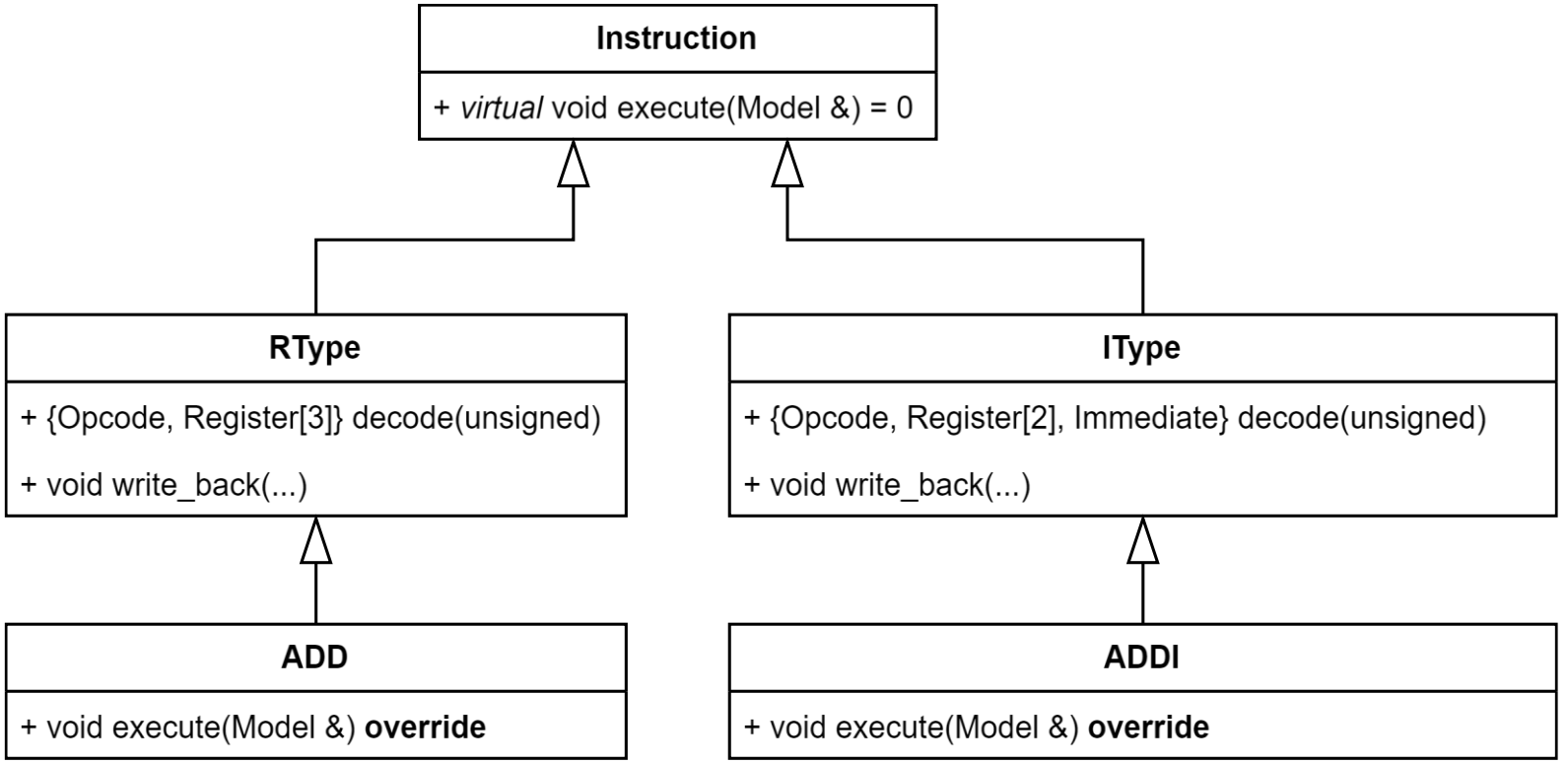
- На этапе **Write back** нужно знать куда записывается результат выполненной инструкции
- При наивном подходе **Decode**, **Execute** и **Write back** сливаются в один блок в switch-case

```
void do_add (inst) {
  auto rd = get_rd(inst);
  auto rs1 = get_rs1(inst);
  auto rs2 = get_rs2(inst);
  write(rd, read(rs1) + read(rs2));
}

switch (inst & mask1) {
  case OP_CLASS_1: {
    switch (inst & mask2) {
      case OP_1:
        do_op1(inst);
        break;
…
}
```

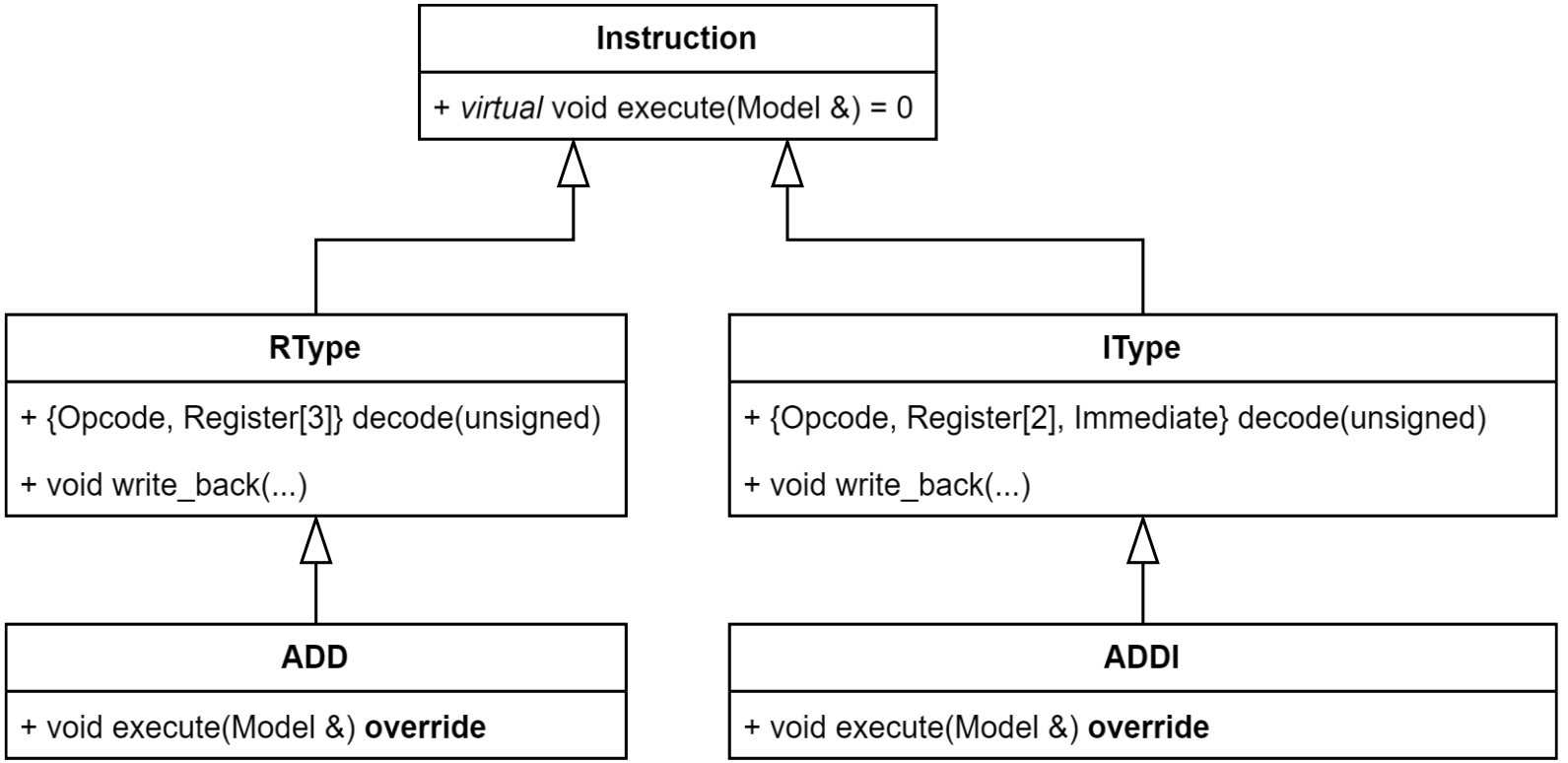
Улучшаем симулятор с ООП

- Распространенный вариант перепроектирования: ввести иерархию классов инструкций



Улучшаем ли?

- Видите ли вы какие-то проблемы в таком подходе?



To be continued …

На следующем занятии

- Рассмотрим проблематику тестирования сложных систем