Генерация тестовых данных для покрытия путей в формальных спецификациях системы команд

Коцыняк Артём Михайлович
kotsynyak@ispras.ru

Научный руководитель:
к.ф.-м.н. Камкин Александр Сергеевич

Схема создания микропроцессора

- Требования

- HDL-описание
  - VHDL, Verilog

- Готовая схема

Верификация микропроцессоров

- Экспертиза
- Формальная верификация
- Имитационное тестирование
  - модульное (сигналы)
  - системное (тестовые программы)

Генерация тестовых программ

- Случайная
- На основе тестовых шаблонов
  - генерация тестовых последовательностей
  - генерация тестовых данных
- На основе моделей

Генерация тестовых данных

- Случайная
- Комбинаторная
- На основе ограничений

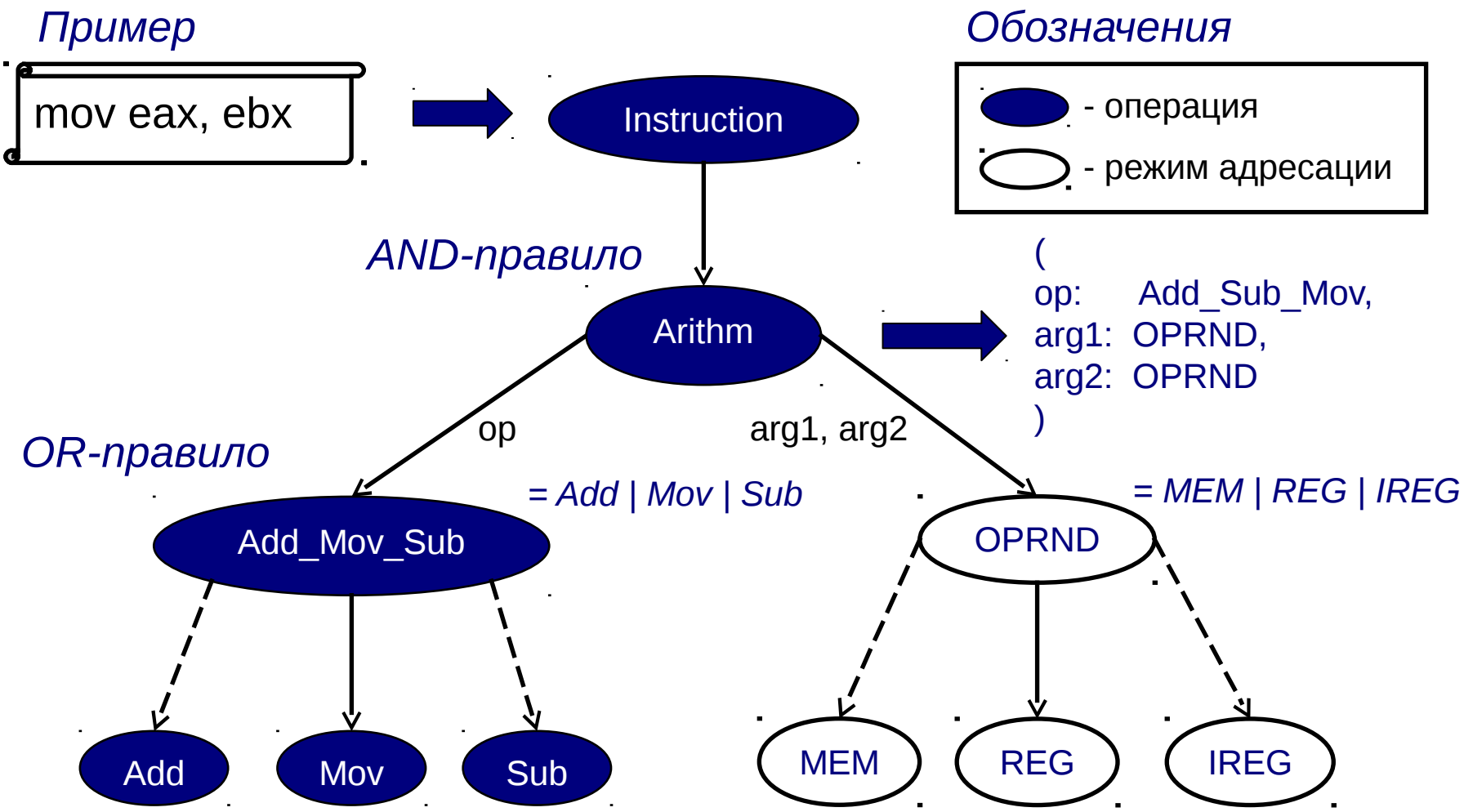
Использование формальных спецификаций системы команд

- Автоматическое извлечение тестовых ситуаций
  - ветви логики операций
  - условия выполнения операций
- Метрики тестового покрытия

Подход MicroTESK к генерации тестовых программ

- Генерация тестов по шаблонам
- Использование модели микропроцессора
- Использование формальных спецификаций для моделирования (nML/Sim-nML)
  - Язык описания архитектуры
  - Система команд описывается иерархической структурой

Структура формализма nML/Sim-nML



Спецификация ресурсов микропроцессора на nML/Sim-nML

Константы:

```
let REGBITS = 5
```

Типы данных:

```
type      bit = card(1)
type   byte_t = card(8)
type halfword = card(16)
type     word = card(32)
type   long_t = int(32)
type  address = card(32)
type    index = card(REGBITS)
```

Память:

```
mem M[2 ** 31, byte_t]
```

Метки:

```
let byte_order = "big"
let PC = "NIA"
let SP = "GPR[29]"
```

Регистры:

```
reg GPR [2 ** REGBITS, long_t]
reg NIA [1, address]
reg LO [1, long_t]
reg HI [1, long_t]
```

Глобальные переменные:

```
mem CIA [1, address]
mem branch [1, bit]
mem JMPADDR [1, address]
```

Спецификация режимов адресации на nML/Sim-nML

Регистры:

```
mode REG(r : index) = GPR[r]
    syntax = format("$%d", r)
    image  = format("%5b", r)
```

Константы:

```
mode IMM16(n : int(16)) = n
    syntax = format("%d", n)
    image  = format("%16b", n)

mode IMM26(n : int(26)) = n
    syntax = format("%d", n)
    image  = format("%26b", n)
```

Спецификация инструкций на nML/Sim-nML

Точка входа:

```
op instruction(op: instr_kind)
    syntax = op.syntax
    image  = op.image
    action = {
        CIA = NIA;
        if  branch == 0 then
            NIA = CIA + 4;
        else
            NIA = JMPADDR;
            branch = 0;
        endif;
        op.action;
        GPR[0] = 0;
    }
```

Группы операций (структура):

```
op instr_kind  = alu_instr
               | load_store_instr
               | branch_instr
               | jump_instr

op alu_instr   = arith_instr
               | logic_instr

op arith_instr = ADD
               | SUB
               | ADDI
               …
op logic_instr = AND
               | OR
               | NOR
               …
```

Спецификация инструкций на nML/Sim-nML

Операция сложения (ADD) :

```
var tmp_unsigned_word [1, card(32)] // Временная переменная
var overflow_bit[1, bit]                    // Временная переменная


op ADD (rd : REG, rs : REG, rt : REG)
    syntax = format ("ADD %s, %s, %s", rd.syntax, rs.syntax, rt.syntax)
    image  = format ("000000%s%s%s%11b", rs.image, rt.image, rd.image, 32)
    action  =  {
        overflow_bit::tmp_unsigned_word = rs + rt;
        if overflow_bit == 1 then
            SignalException("Integer Overflow Exception");
        else
            rd = tmp_unsigned_word;
        endif;
    }
```

Спецификация инструкций на nML/Sim-nML

Операция условного перехода (BEQ):

```
var tmp_signed_word [1, int(32)]  // Временная переменная


op BEQ (rs : REG, rt : REG, offset : IMM16)
    syntax = format ("BEQ %s, %s, %s", rs.syntax, rt.syntax, offset.syntax )
    image = format ("000100%s%s%s", rs.image, rt.image, offset.image )
    action = {
        if  rs == rt  then
            branch = 1;
            tmp_signed_word = offset;
            tmp_signed_word = tmp_signed_word << 2;
            JMPADDR = NIA + tmp_signed_word;
        endif;
    }
```

Спецификация инструкций на nML/Sim-nML

Операция подсчёта ведущих нулей (CLZ):

```
var tmp_signed_byte [1, int(8)]  // Временная переменная

op CLZ(rd : index, rs : REG)
    syntax = format (“CLZ %d, %s", rd, rs.syntax)
    image  = format (“011100%s%5b%5b%11b", rs.image, rd, rd, 33)
    action  =  { tmp_signed_byte = 31; GPR [rd] = 32; loop; }
    loop     =  {
        if tmp_signed_byte >= 0 then
            if rs<tmp_signed_byte..tmp_signed_byte> == 0 then
                tmp_signed_byte = tmp_signed_byte - 1;
            else
                GPR [rd] = 31 - tmp_signed_byte;
                tmp_signed_byte = -1;
            endif;
            loop;
     endif;
}
```

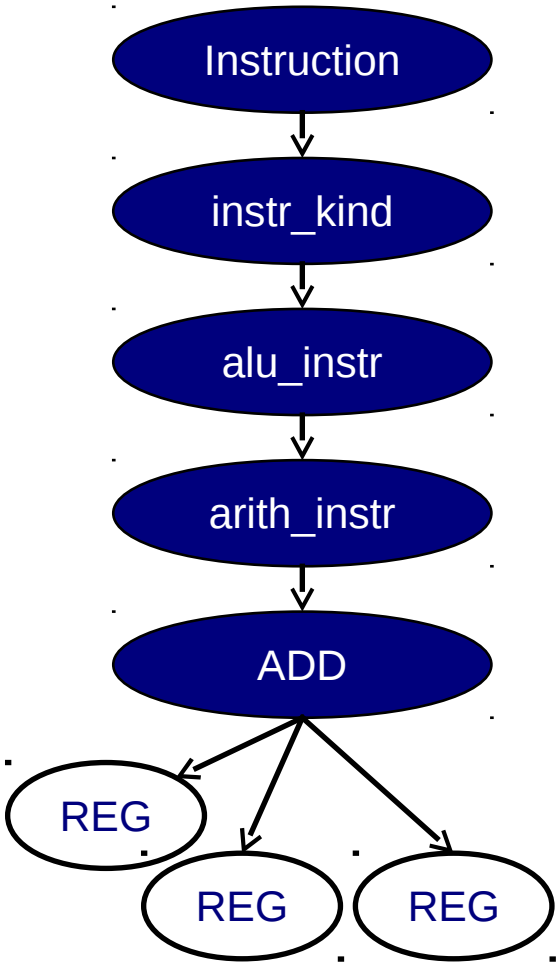
Свойства кода спецификации

- Простые условия ветвления
- Циклы с небольшим максимальным числом итераций
- Пути в коде являются тестовыми ситуациями

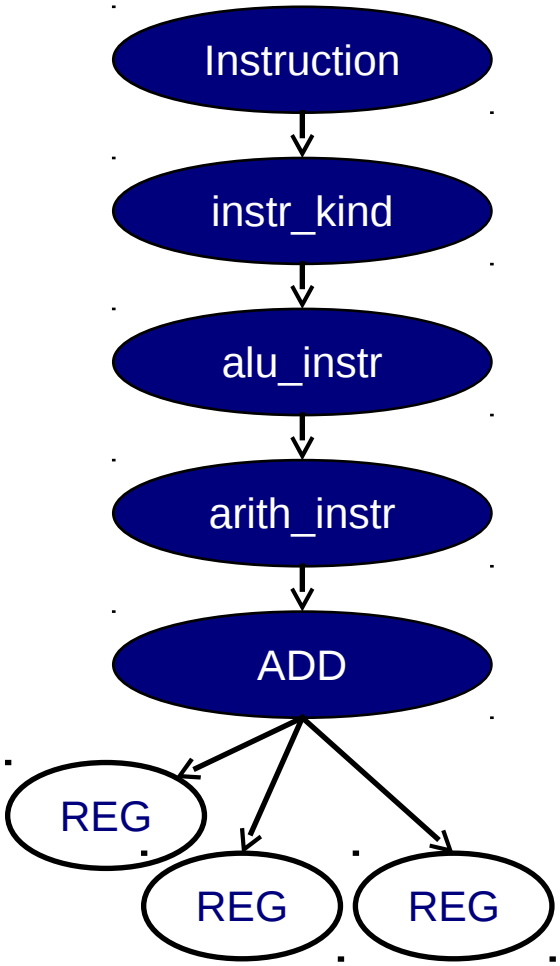
Извлечение путей выполнения из спецификации системы команд

- Построение полного кода операции
  - обход иерархии инструкций
  - определение входных данных
  - преобразование режимов адресации
  - раскрутка циклов
- Построение предикатов для путей
- Определение побочных эффектов

Обход иерархии инструкций

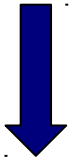


Определение входных данных



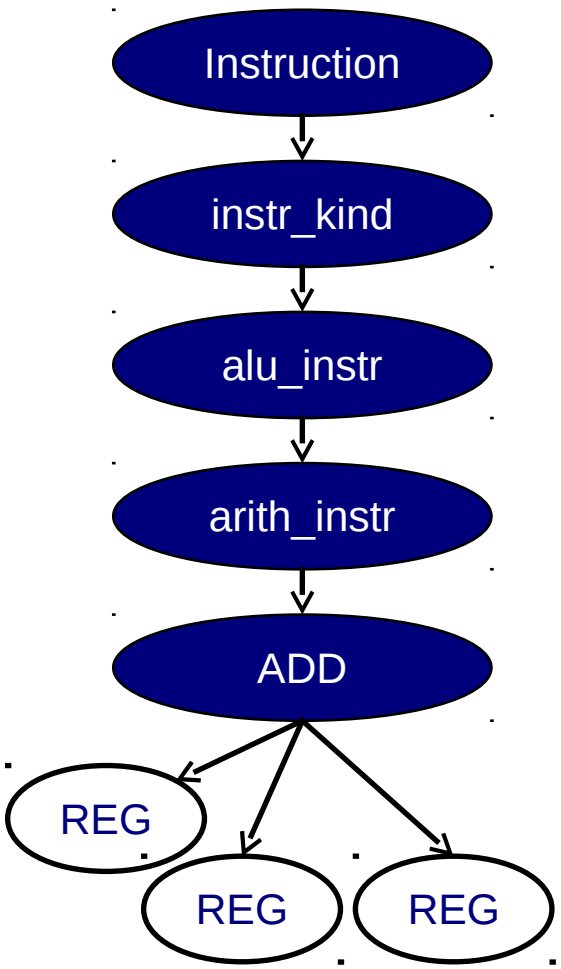

```
op ADD (rd : REG, rs : REG, rt : REG)
image = format ("000000%s%s%s%11b",
    rs.image, rt.image, rd.image, 32)

mode REG(r : index) = GPR[r]
image = format("%5b", r)
```



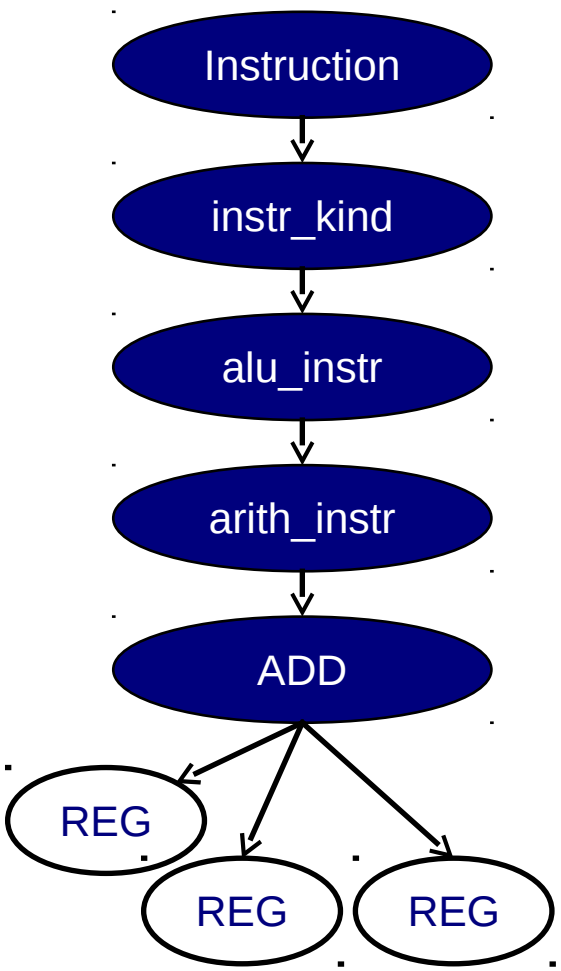
```
op ADD (rd : index, rs : index, rt : index)
image = format ("000000%5b%5b%5b%11b",
    rs, rt, rd, 32)
```

Построение полного кода инструкции




```
op ADD (rd : index, rs : index, rt : index)
action = {
CIA = NIA;
if branch == 0 then
    NIA = CIA + 4;
else
    NIA = JMPADDR;
    branch = 0;
endif;
overflow_bit::tmp_unsigned_word = GPR[rs] + GPR[rt];
if overflow_bit == 1 then
    SignalException("Integer Overflow Exception");
else
    GPR[rd] = tmp_unsigned_word;
endif;
GPR[0] = 0;
}
```

Построение предикатов для путей




```
op ADD (rd : index, rs : index, rt : index)
action = {
CIA = NIA;
if branch == 0 then
    NIA = CIA + 4;
else
    NIA = JMPADDR;
    branch = 0;
endif;
overflow_bit::tmp_unsigned_word = GPR[rs] + GPR[rt];
if overflow_bit == 1 then
    SignalException("Integer Overflow Exception");
else
    GPR[rd] = tmp_unsigned_word;
endif;
GPR[0] = 0;
}
```

Построение предикатов для путей

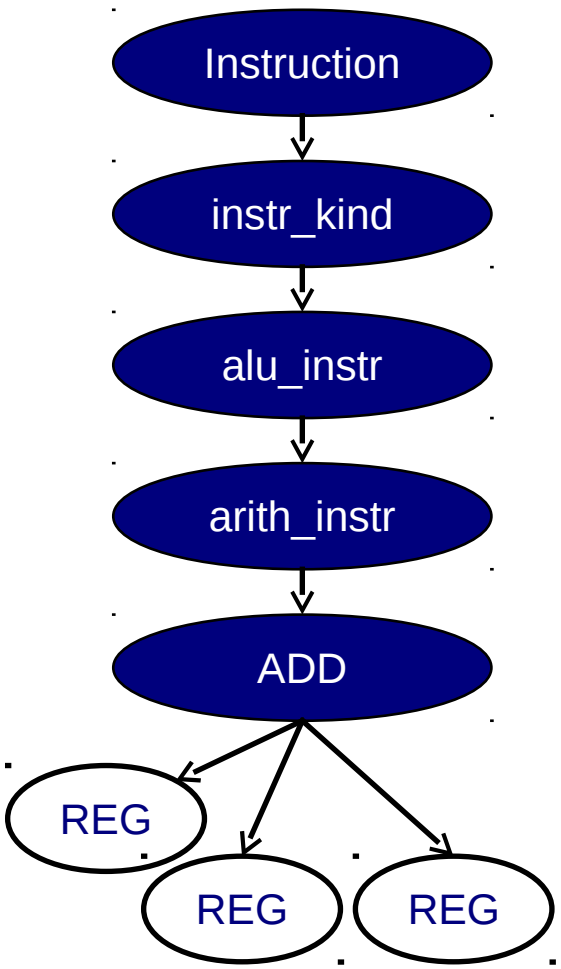
$branch_0 = 0 \wedge overflow\_bit_0 :: tmp\_unsigned\_word_0 = GPR[rs]_0 + GPR[rt]_0 \wedge overflow\_bit_0 = 1$

$branch_0 \neq 0 \wedge overflow\_bit_0 :: tmp\_unsigned\_word_0 = GPR[rs]_0 + GPR[rt]_0 \wedge overflow\_bit_0 = 1$

$branch_0 = 0 \wedge overflow\_bit_0 :: tmp\_unsigned\_word_0 = GPR[rs]_0 + GPR[rt]_0 \wedge overflow\_bit_0 \neq 1$

$branch_0 \neq 0 \wedge overflow\_bit_0 :: tmp\_unsigned\_word_0 = GPR[rs]_0 + GPR[rt]_0 \wedge overflow\_bit_0 \neq 1$

Определение побочных эффектов




```
op ADD (rd : index, rs : index, rt : index)
action = {
CIA = NIA;
if branch == 0 then
    NIA = CIA + 4;
else
    NIA = JMPADDR;
    branch = 0;
endif;
overflow_bit::tmp_unsigned_word = GPR[rs] + GPR[rt];
if overflow_bit == 1 then
    SignalException("Integer Overflow Exception");
else
    GPR[rd] = tmp_unsigned_word;
endif;
GPR[0] = 0;
}
```

Определение побочных эффектов

$branch_0 = 0 \wedge overflow\_bit_0 :: tmp\_unsigned\_word_0 = GPR[rs]_0 + GPR[rt]_0 \wedge overflow\_bit_0 = 1 \Longrightarrow NIA_1 = NIA_0 + 4 \wedge GPR[rd]_1 = tmp\_unsigned\_word_0 \wedge GPR[0]_1 = 0$

$branch_0 \neq 0 \wedge overflow\_bit_0 :: tmp\_unsigned\_word_0 = GPR[rs]_0 + GPR[rt]_0 \wedge overflow\_bit_0 = 1 \Longrightarrow NIA_1 = JMPADDR_0 \wedge branch_1 = 0 \wedge GPR[rd]_1 = tmp\_unsigned\_word_0 \wedge GPR[0]_1 = 0$

$branch_0 = 0 \wedge overflow\_bit_0 :: tmp\_unsigned\_word_0 = GPR[rs]_0 + GPR[rt]_0 \wedge overflow\_bit_0 \neq 1 \Longrightarrow NIA_1 = NIA_0 + 4 \wedge exception(\text{"Integer Overflow Exception"}) \wedge GPR[0]_1 = 0$

$branch_0 \neq 0 \wedge overflow\_bit_0 :: tmp\_unsigned\_word_0 = GPR[rs]_0 + GPR[rt]_0 \wedge overflow\_bit_0 \neq 1 \Longrightarrow NIA_1 = JMPADDR_0 \wedge branch_1 = 0 \wedge exception(\text{"Integer Overflow Exception"}) \wedge GPR[0]_1 = 0$

Стратегии генерации тестовых данных

- Несколько наборов данных для заданной тестовой последовательности
- Набор данных с максимальным покрытием путей в данной тестовой последовательности
- Отслеживание общего покрытия путей в наборе тестовых последовательностей

Генерация тестовых данных

- Построение ограничений шаблона
  - Поиск путей с требуемыми эффектами
- Использование информации о покрытии
  - Поиск нерассмотренных выполнимых путей
  - Ограничения путей добавляются к ограничениям шаблона
- Использование решателей ограничений для генерации данных

Проблемы

- Определение побочных эффектов
  - nML/Sim-nML var vs. mem
  - Вызов внешних функций
- Внутренние механизмы процессора
  - Подсистема управления памятью
  - Модуль управления переходами
  - Конвейер

Дальнейшие планы

- Использование модели конвейера
- Создание базы тестовых знаний
  - Объединение подходов к генерации данных
  - Переиспользование тестового знания

Спасибо за внимание!