MicroTESK: Среда генерации тестовых программ для микропроцессоров, основанная на формальных спецификациях системы команд

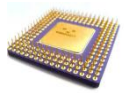
Андрей Татарников
andrewt@ispras.ru

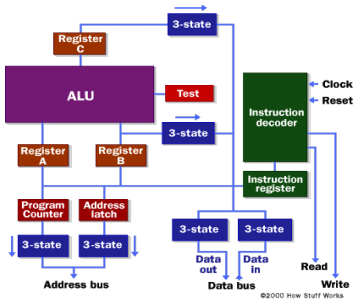
Научный руководитель
к.ф.-м.н. Александр Камкин

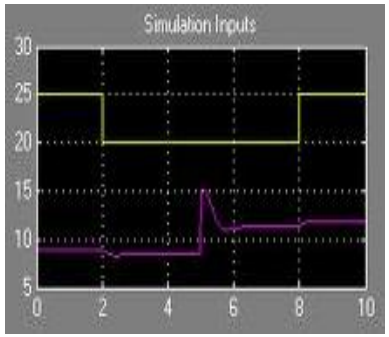
ISP RAS

Institute for System Programming of the Russian Academy of Sciences (ISPRAS)
http://hardware.ispras.ru

Создание микропроцессоров







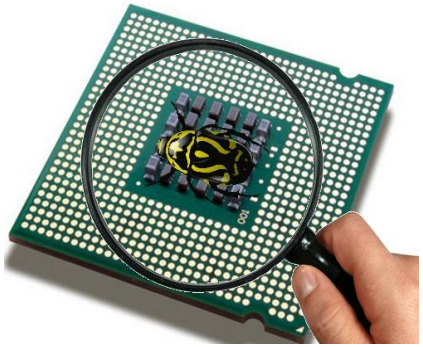
Языки описания цифровой аппаратуры (HDL)

- Verilog
- VHDL

Синтез цифровой аппаратуры

- RTL модель (HDL)
- Netlist (логические примитивы)
- Шаблон для производства чипа

Верификация микропроцессоров

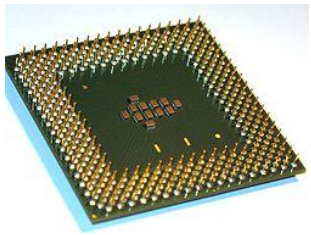


Применяется к RTL-моделям и включает

- Экспертизу (code review)
- Формальную верификацию
- Имитационное тестирование
  - Модульное (сигналы)
  - Системное (тестовые программы)

Системное тестирование

Симуляция и сравнение трасс



Тестовые программы

```
lui s1,0x27
ori s1,s1,0xc8
lui s3,0x4e
ori s3,s3,0xf7
...
```

RTL-модель дизайна (Verilog, VHDL)

Эталонная модель (эмулятор) на C/C++

Сравнение трасс

✕ **Возникла ошибка**

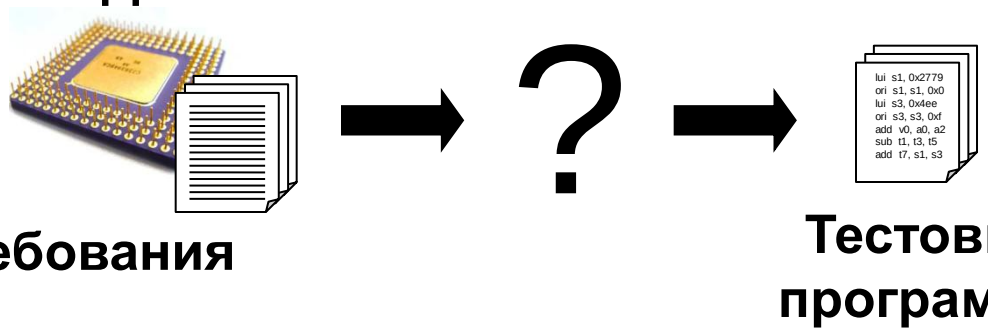
☑ **Успешный прогон**

Системное тестирование

Тесты со встроенными проверками (self-checking tests)

- Эталонная модель включена в генератор
- Инструкции симулируются при генерации
- Ожидаемые результаты включаются в тесты

Создание тестовых программ



Применяемые способы

- Кросс-компиляция существующего ПО
- Ручная разработка
- Случайная генерация
- Генерация на основе на тестовых шаблонов
- Генерация на основе на моделей

Основные виды автоматизированных тестов

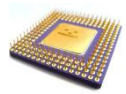
```
lui s1,0x27
ori s1,s1,0xc8
lui s3,0x4e
ori s3,s3,0xf7
...
```

- Случайные
- Комбинаторные
- Направленные

Проблемы автоматических генераторов

- Настройка под новую архитектуру
- Интеграция новых методов генерации
- Повторное использование тестового знания

Решение: использование моделей микропроцессора



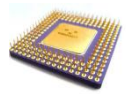
Основные компоненты генератора

- Ядро генератора
- Модель микропроцессора
  - Модель архитектуры
  - Модель покрытия (тестовое знание)

Примеры генераторов

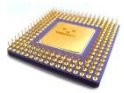
- RAVEN (ARM, изначально Obsidian Software)
- Genesys-Pro (IBM Research)
- MicroTESK (ISPRAS)

Основные характеристики среды MicroTESK



- Генерирует тесты по шаблонам
- Использует модель микропроцессора
- Использует формальные спецификации для моделирования (nML/Sim-nML)
- Состоит из среды моделирования и среды генерации тестов
- Извлекает данные о тестовом покрытии из спецификаций
- Позволяет интегрировать новые методы генерации тестов

Структура MicroTESK



Формальные спецификации

Инженер-верификатор

Тестовые шаблоны

Среда моделирования

Транслятор

- Генератор модели архитектуры
- Генератор тестового знания

Библиотеки

- Библиотека модели архитектуры
- Библиотека тестового знания

Модель микропроцессора

- Модель архитектуры
- Тестовое знание

Тестовые программы

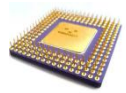
Среда генерации тестов

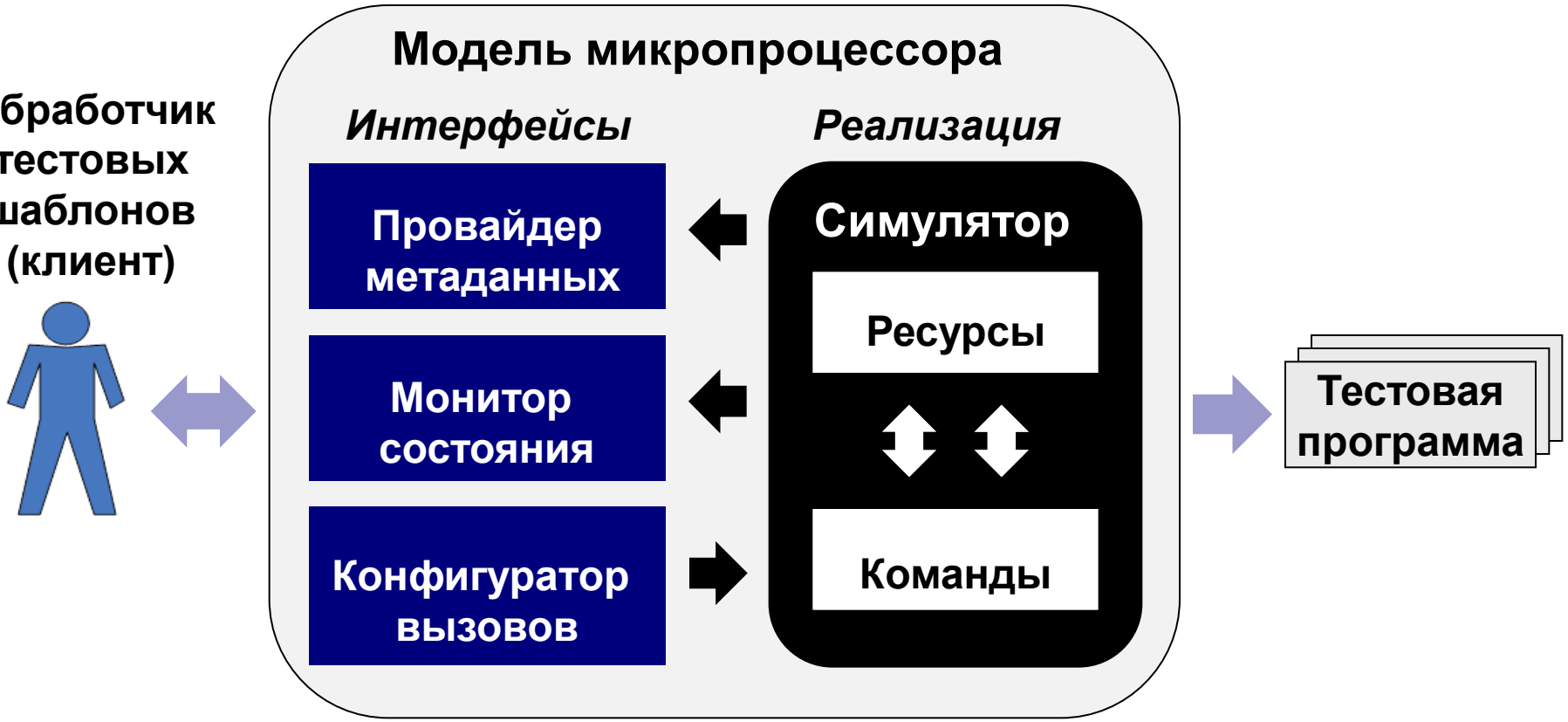
- Обработчик тестовых шаблонов
- Решатель ограничений

Библиотека генераторов

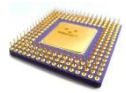
- Генераторы тестовых последовательностей
- Генераторы тестовых данных

Концепция модели архитектуры





Формальные спецификации

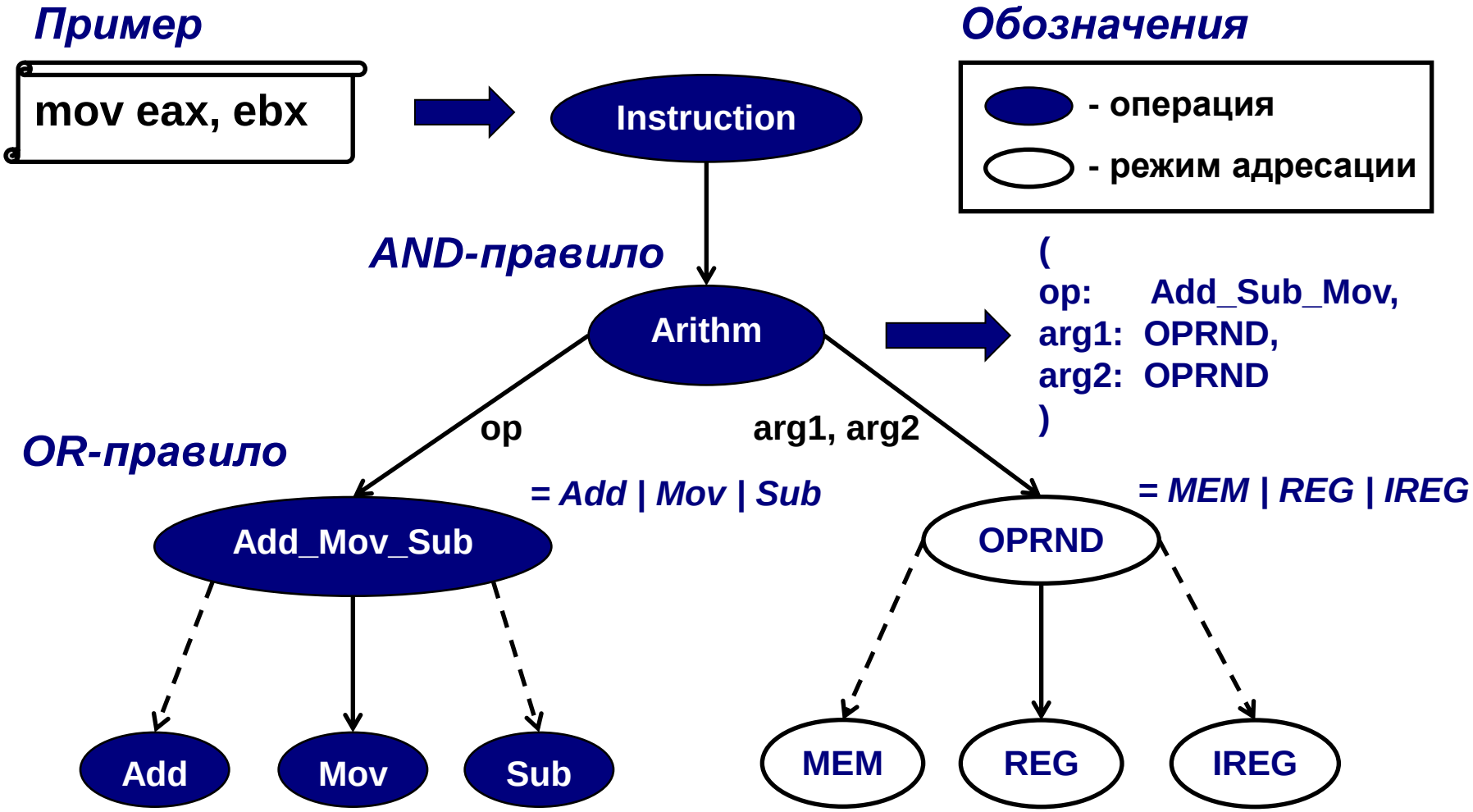


Общие сведения об nML/Sim-nML

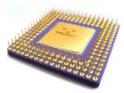
- Язык описания архитектуры (Architecture Description Language, ADL)
- Разработан в TU Berlin в начале 90’х
- Описывает систему команд микропроцессора как иерархическую структуру
- Основан на атрибутной грамматике
- Применяется для создания симуляторов и генераторов кода

Обзор языка nML/Sim-nML

Структура формализма



Обзор языка nML/Sim-nML



Спецификация ресурсов микропроцессора

Константы и метки:

```
let  MSIZE = 2 ** 6
let  REGS  = 16
let  byte_order = ”little”
let  SP = ”GPR[13]”
```

Типы данных:

```
type  index  = card(6)
type  nibble = card(4)
type  byte_t = int(8)
```

Регистры:

```
reg  R[REGS, byte_t]
reg  PC[1, byte_t]
```

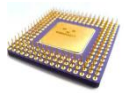
Память:

```
mem  M[MSIZE, byte_t]
```

Переменные:

```
var  temp[1, byte_t]
var  shifter_carry_out[1, card(1)]
var  ALU_OUT[1, card(32)]
```

Обзор языка nML/Sim-nML



Спецификация команд
Операции и режимы адресации

Режимы (правила AND и OR):

```
mode REG(i: nibble) = R[i]
syntax = format("R%d", i)
image  = format("01%4b", i)

mode OPRND = MEM | REG | IREG
```

Операции (правила AND и OR):

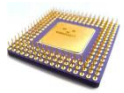
```
op Add(dest: OPRND, src: OPRND)
syntax = "add"
image  = "00"
action  = { dest = dest + src; }

op Add_Sub_Mov = Add | Sub | Mov
```

Корневая операция (точка входа):

```
op instruction(
    child: Add_Sub_Mov)
syntax = child.syntax
image  = child.image
action  = {
    child.action;
    PC = PC + 2;
}
```

Пример спецификации MIPS



Спецификация ресурсов

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]"
```

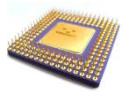
Регистры:

```
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]
var branch [1, bit]
```

Пример спецификации MIPS



Спецификация режимов адресации

Регистры (для инструкций R):

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

Пример использования (nML):

```
op ADD (rd: REG, rs: REG, rt: REG)
```

Код Ruby шаблона:

```
add reg(0), reg(1), reg(2)
```

Константы (для инструкций I):

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

Пример использования (nML):

```
op ADDI (rt: REG, rs: REG, i: IMM16)
```

Код Ruby шаблона:

```
addi reg(0), reg(1), imm16(20)
```

Константы (для инструкций J):

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

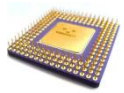
Пример использования (nML):

```
op J (target : IMM26)
```

Код Ruby шаблона:

```
j imm26(0x1000000)
```

Пример спецификации MIPS



Спецификация операций (часть 1)

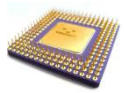
Точка входа:

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

Пример спецификации MIPS



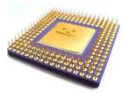
Спецификация операций (часть 2)

Операция сложения (ADD) :

```
var tmp_unsigned_word [1, card(32)]  // 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%s00000100000",
                     rs.image, rt.image, rd.image)
    action  =  {
        overflow_bit::tmp_unsigned_word = rs + rt;
        if overflow_bit == 1 then
            SignalException("Integer Overflow Exception");
        else
            rd = tmp_unsigned_word;
        endif;
    }
```

Пример спецификации MIPS



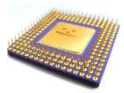
Спецификация операций (часть 3)

Операция условного перехода (BEQ):

```
var tmp_signed_word [1, int(32)]  // Временная переменная
var JMPADDR [1, address]          // 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;
    }
```

Пример спецификации MIPS

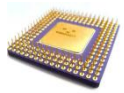


Спецификация операций (часть 4)

Операция сравнения значения регистра и константы (SLTI):

```
op SLTI (rt : REG, rs : REG, imm : IMM16)
    syntax = format ("SLTI %d, %s, %s", rt.syntax, rs.syntax, imm.syntax)
    image  = format ("001010%s%s%s", rs.image, rt.image, imm.image)
    action  = {
        if  rs < imm then
            rt = 1;
        else
            rt = 0;
        endif;
    }
```

Пример спецификации MIPS



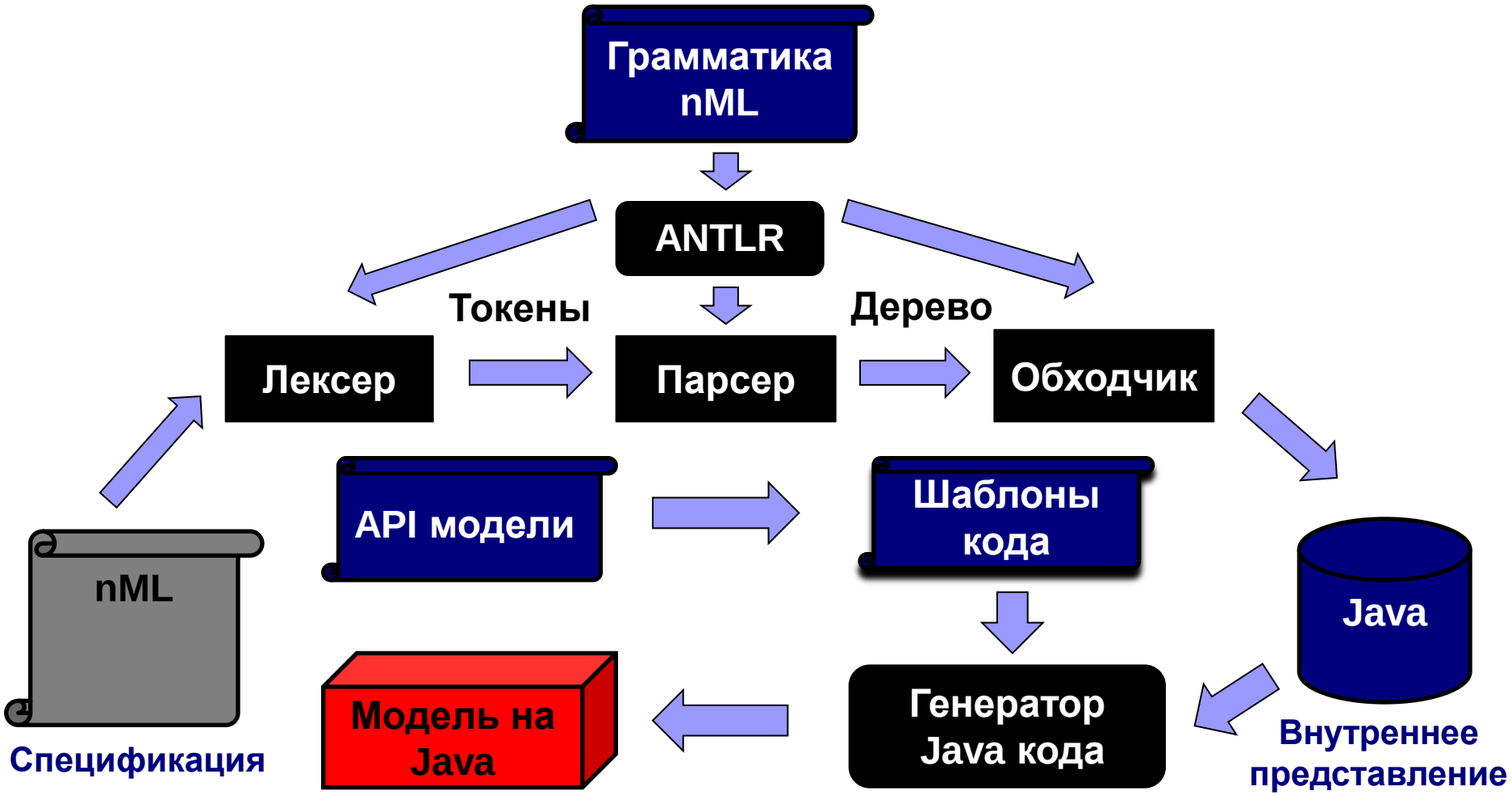
Спецификация операций (часть 5)

Операция безусловного перехода (J):

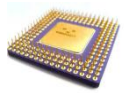
```
op J (target : IMM26)
    syntax = format ("J %s", target.syntax)
    image  = format ("000010%s", target.image)
    action  = {
        branch = 1;
        CIA = NIA;
        JMPADDR = CIA & 0xF0000000;
        JMPADDR = JMPADDR | ( target << 2);
    }
```

Транслятор MicroTESK

Схема трансляция спецификации



Анализ кода



Извлечение информации о покрытии

```
op ADD(rd: GPR, rs: GPR, rt: GPR)
action = {
    if(NotWordValue(rs) || NotWordValue(rt)) then
        UNPREDICTABLE();
    endif;
    tmp_word = rs<31..31>::rs<31..0> + rs<31..31>::rt<31..0>;
    if(tmp_word<32..32> != tmp_word<31..31>) then
        SignalException("IntegerOverflow");
    else
        rd = sign_extend(tmp_word<31..0>);
    endif;
}

syntax = format("add %s, %s, %s",  rd.syntax, rs.syntax, rt.syntax)

op ALU = ADD | SUB | ...
```

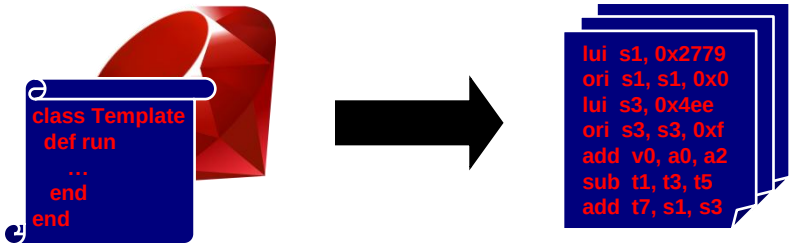
Предусловие

Тестовые ситуации

Классы эквивалентности

Тестовые шаблоны

Генерация на основе шаблонов



Основные свойства тестовых шаблонов

- Абстрактное описание теста
- Высокоуровневый язык описания (Ruby)
- Расширяемая среда
- Предусловия и постусловия
- Тестовые ситуации и ограничения
- Методы составления последовательностей
- Интеграция различных методов генерации

Генерация тестов по шаблонам

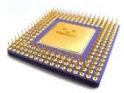


Схема генерации

Шаблон на Ruby:

```
block {
    eor r(0), r(0), register0
    add reg(2), reg(1), register0 do normal end
    mov_immediate reg(1), immediate(0, 0xFF)
    add reg(1), reg(4), register4 do random end
    sub reg(3), reg(2), register1 do overflow end
}
```

Генерация тестовых последовательностей

Генерация тестовых данных

Симуляция на модели

Инициализатор →

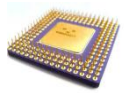
Тестовый код →

Ассемблерный код:

```
EOR R1, R1, R1
MOV R1, R1, LSL #8
ADD R1, R1, #df<<<0*2
MOV R1, R1, LSL #8
ADD R1, R1, #56<<<0*2
MOV R1, R1, LSL #8
ADD R1, R1, #bf<<<0*2
MOV R1, R1, LSL #8
ADD R1, R1, #ef<<<0*2
```

```
EOR R0, R0, R0
ADD R2, R1, R0
MOV R1, #ff<<<0*2
ADD R1, R4, R4
SUB R3, R2, R1
```

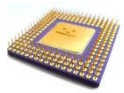
Язык описания тестовых шаблонов



Пример простого шаблона

```
class TestCase01 < Template
  def pre # Выполняется перед основным кодом (инициализирующий код)
    addi reg(17), reg(0), imm16(5)
    addi reg(18), reg(0), imm16(10)
    addi reg(19), reg(0), imm16(7)
  end
  def run # Основной код тестового примера
    add reg(8), reg(17), reg(18)
    add reg(9), reg(19), reg(20)
    sub reg(16), reg(8), reg(9)
  end
  def post # Выполняется после основного кода
    sw reg(16), reg(21), imm16(128)
  end
end # class TestCase01
```

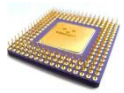
Генерация тестовых последовательностей



Блоки инструкций

```
class TestCase01 < Template
  def run
    block (:combine => “product”, :compose => “random”) {
      block (:engine => “random”, :length => 3, :count => 2) {  # Nested Block A
        add  reg(8),  reg(16), reg(17)
        sub  reg(9),  reg(18), reg(19)
        and reg(10), reg(16), reg(17)
        or    reg(11), reg(18), reg(19)
      }
      block (:engine => "permutate") {  # Nested Block B
        lw reg(12), reg(20), imm16(0)
        sw  reg(9), reg(21), imm16(0)
      }
    }
  end
end
```

Генерация тестовых последовательностей



Сгенерированный код

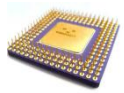
```
# Combination (1, 1)
sub $9, $18, $19    # Block A
lw $12, 0($20)      # Block B
or $11, $18, $19    # Block A
sw $9, 0($21)       # Block B
add $8, $16, $17    # Block A

# Combination (1, 2)
sw $9, 0($21)       # Block B
sub $9, $18, $19    # Block A
lw $12, 0($20)      # Block B
or $11, $18, $19    # Block A
add $8, $16, $17    # Block A

# Combination (2 , 1)
and $10, $16, $17   # Block A
and $10, $16, $17   # Block A
lw $12, 0($20)      # Block B
add $8, $16, $17    # Block A
sw $9, 0($21)       # Block B

# Combination (2, 2)
and $10, $16, $17   # Block A
sw $9, 0($21)       # Block B
and $10, $16, $17   # Block A
lw $12, 0($20)      # Block B
add $8, $16, $17    # Block A
```

Генерация тестовых последовательностей



Алгоритмы создания последовательностей

Типы генераторов

- Комбинаторы (комбинирование инструкций)
- Композиторы (объединение последовательностей)

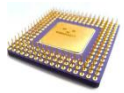
Стандартные комбинаторы

- Random
- Product
- Diagonal

Стандартные композиторы

- Random
- Catenation
- Nesting

Генерация тестовых данных



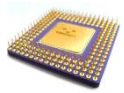
Тестовые ситуации

- Целевые состояния
- Целевые ветви логики
- Условия достижения/выполнения

Ограничения

- Предусловия
- Интервалы возможных значений
- Зависимости между входными данными
- Псевдослучайные значения

Генерация тестовых данных



Тестовые ситуации в шаблонах

- Ассоциируются с отдельными инструкциями
- Описывают условия перехода в определённое состояние / выполнение определённой ветви логики
- Возможно комбинирование тестовых ситуаций
- Возможно задавать вероятностное распределение

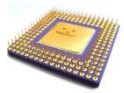
Пример:

```
class TestCase01 < Template
  def run
    add reg(8), reg(16), reg(17) do overflow([0.5]) end
    add reg(9), reg(18), reg(19) do normal end
  end
end
```

Вызывает целочисленное переполнение с вероятностью 50%

Гарантирует выполнение без переполнения

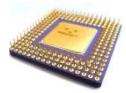
Описание тестовых ситуаций



Описание и решение ограничений

- Библиотека Fortress (ISPRAS), http://forge.ispras.ru/projects/solver-api/
  - Библиотечные классы
  - XML
- Язык SMT-LIB, http://www.smtlib.org/
- Решатель Z3 (Microsoft Research), http://z3.codeplex.com/
- Решатель Yices, http://yices.csl.sri.com/

Заключение

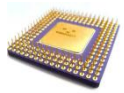


Достигнутые результаты

- Прототип MicroTESK
  - Транслятор языка Sim-nML
  - Язык описания тестовых шаблонов
- Модель архитектуры ARM
- Модель архитектуры MIPS

Дальнейшие планы

- Поддержка Elbrus
- Моделирование системы управления памятью
- Извлечение тестового покрытия



Спасибо!
Вопросы?