Генератор тестовых программ для архитектуры ARMv8 на основе инструмента MicroTESK

Александр Камкин, Артем Коцыняк

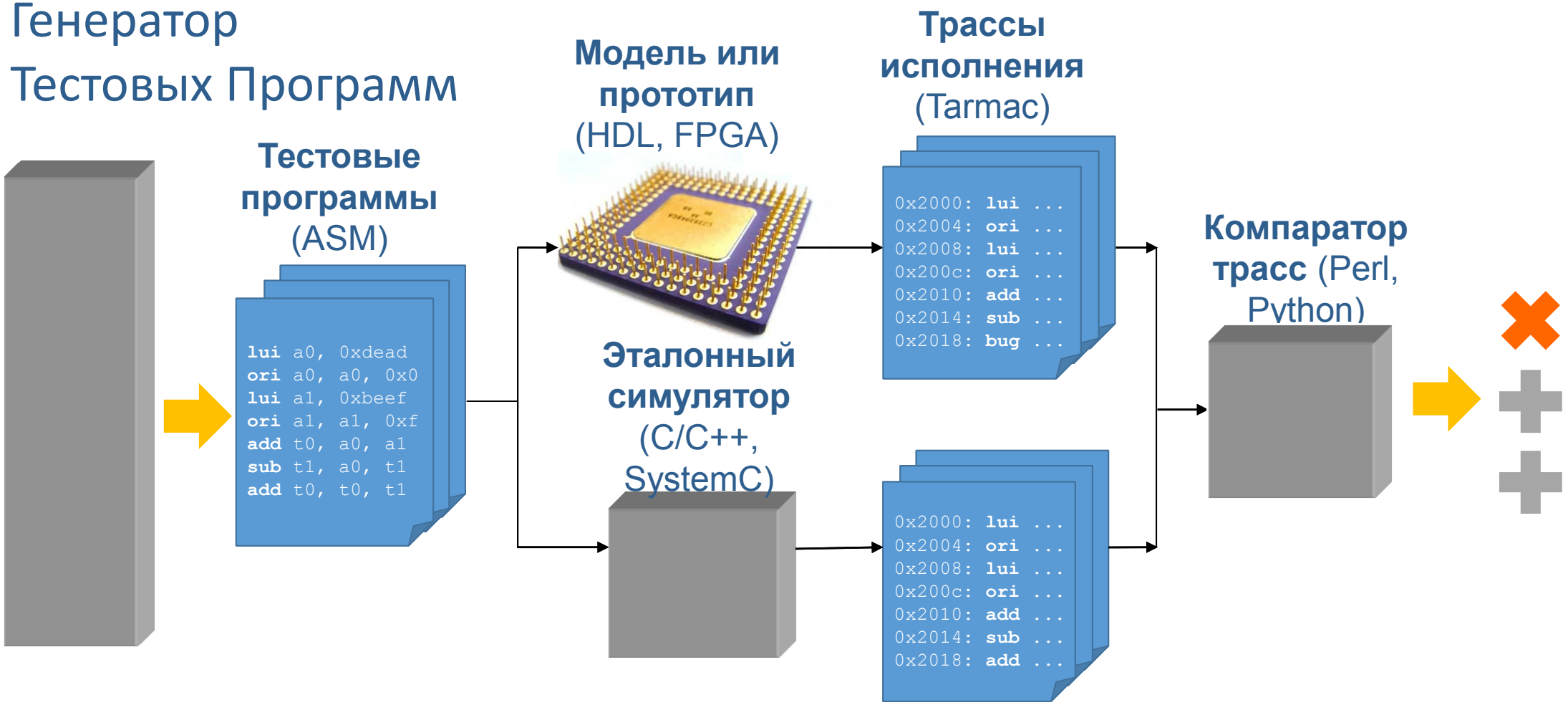
Александр Проценко, Андрей Татарников, Михаил Чупилко

ИСП РАН

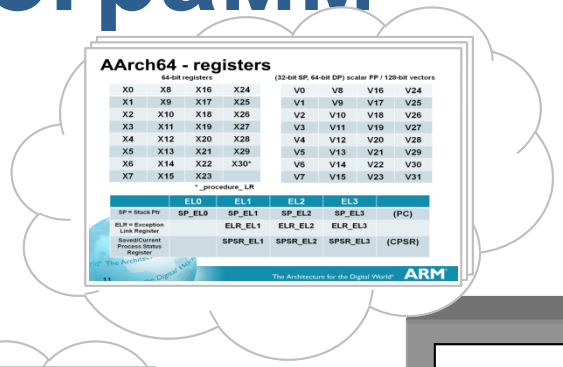
Институт системного программирования Российской академии наук (ИСП РАН)
109004, г. Москва, ул. Александра Солженицына, д. 25 | http://www.ispras.ru

Введение Тестирование микропроцессоров

Генератор Тестовых Программ



MicroTESK – Генератор тестовых программ



Параметры генерации

- Последовательности команд
- Ограничения на данные
- Распределения вероятностей
- и т.д. и т.п.

Инженеры-Верификаторы

Генератор на базе **MicroTESK**

Транслятор → Модель процессора

Спецификации

Тестовые шаблоны → Ядро генератора

Тестовые программы

```
lui a0, 0xdead
ori a0, a0, 0x0
lui a1, 0xbeef
ori a1, a1, 0xf
add t0, a0, a1
sub t1, a0, t1
add t0, t0, t1
```

Язык nML: Типы, Регистры и Режимы Доступа

```
// Типы Данных
type BYTE  = card(8)   // Unsigned
type SHORT = int(16)   // Signed
type DWORD = card(64)  // Unsigned
...

// Memory Operations
let MemOp_LOAD  = 0 // Int Constant
let MemOp_STORE = 1

// Типы Доступа к Памяти
let AccType_NORMAL  = 0
let AccType_ORDERED = 5
let AccType_PTW     = 8
...
```

1

```
// Регистры и Псевдонимы
reg X[32, DWORD] // 32 DWORD Registers
reg SP[DWORD] alias = X[31]
...

// Режимы Доступа к Регистрам
mode REG(i: card(5)) = X[i]
  syntax = format("x%d", i) // E.g. x13
  image  = format("%5s", i) // E.g. 01101
...
```

2

```
mem MEM[(2 ** 48) / 8, DWORD]
```

3

Язык nML

Операции и их Группы

```
// Инструкция чтения данных
op LDAR(rt: REG, rn: REG)
  syntax = // Ассемблерный Формат
    format("ldar %s, [%s, #0]",
      rt.syntax,
      if rn.i == 31
        then "sp"
        else rn.syntax
      endif)
  image = // Двоичный Формат
    format("11001000110111111111111%5s%5s",
      rn.image,
      rt.image)
  action = { // Семантика Инструкции
    F(MemOp_LOAD, rt, rn, 64,
      AccType_ORDERED).action;
  }
```

1

```
op F(opc: BYTE, rt: REG, rn: REG, ...)
  action = {
    va = rn; // Register Read
    if opc == MemOp_LOAD then
      if va<...> != 0 then
        exception("AlignmentFault");
      endif;
      [redacted]
      temp = MEM[va >> 3];
      rt   = temp<...>; // Register Write
      ...
    endif;
  }
```

2

```
// Группа Операций
op LOAD = LDR | LDUR | LDAR | ...
```

3

Язык mmu? Типы адресов и Сегменты

```
address VA(
  vaddress     : 64, // Virtual Address
  acctype      : 4,  // Memory Access Type
  iswrite      : 1,  // Store/Load
  wasaligned   : 1,  // Aligned/Unaligned
  size         : 6   // Data Portion Size
)

address IPA(
  ipaddress    : 48, // Intermediate Address
  level        : 2,  // Page Table Level
  ...
  secondstage  : 1,  // Second/First Stage
)

address PA(
  paddress     : 48, // Physical Address
  ...
)
```

1

```
// Mapping of VA to PA
segment SEG(va: VA) = (pa: PA)
  range = (0, 0x0000ffffFFFFffff)
  read  = {
    // Flat Address Translation
    pa.paddress = va.vaddress<47..0>;
    ...
    // Or Table-Based Translation
    if [redacted] then
      record = TLB(va);
      ...
      pa.paddress = f(record, va)
    endif;
  }
```

2

```
buffer TLB(va: VA)
  entry = (...) // TLB Record Format
  ...           // Other Parameters
```

3

Язык mmu… Буферы Трансляции Адресов

```
// Register-Mapped Buffer
register buffer TLB(va: VA)
  ways    = N      // Associativity
  sets    = M      // Number of Sets
  tag     = f(va)  // Tag Function
  index   = g(va)  // Index Function

  policy = none    // Eviction Policy

  entry   = (      // Entry Format
    perms       : Permissions,
    nG          : 1,
    contiguous  : 1,
    blocksize   : 32,
    addrdesc    : AddrDesc
  )
```

1

```
// Memory-Mapped Buffer
[redacted] buffer TranslationTable(va: VA)
  entry = (
    IGNORED     : 9,  // <63..55>
    XN          : 1,  //      <54>
    PXN         : 1,  //      <53>
    Contiguous  : 1,  //      <52>
    RES0        : 4,  // <51..48>
    pa          : 36, // <47..12>
    nG          : 1,  //      <11>
    AF          : 1,  //      <10>
    SH          : 2,  //    <9..8>
    AP          : 2,  //    <7..6>
    NS          : 1,  //       <5>
    AttrIndx    : 3,  //    <4..2>
    page        : 1,  //       <1>
    valid       : 1   //       <0>
) ...
```

2

Язык mmu: Логика Управления

```
mmu MMU(va: VA) = (data: 64)
  read = {            // Load Operation
    if va.acctype != AccType_PTW then
      pa = AArch64TranslateAddress(va);
    else
      pa = SEG(va); // Flat Translation
    endif;
    if L1(pa).hit then ... else ...
      if va.acctype != AccType_PTW then
        temp = AArch64MemSingleRead(va);
      else
        temp = MEM(pa);
      endif;
    ...
    endif;
    data = temp<...>;
  }
  write = { ... }    // Store Operation
}
```

1

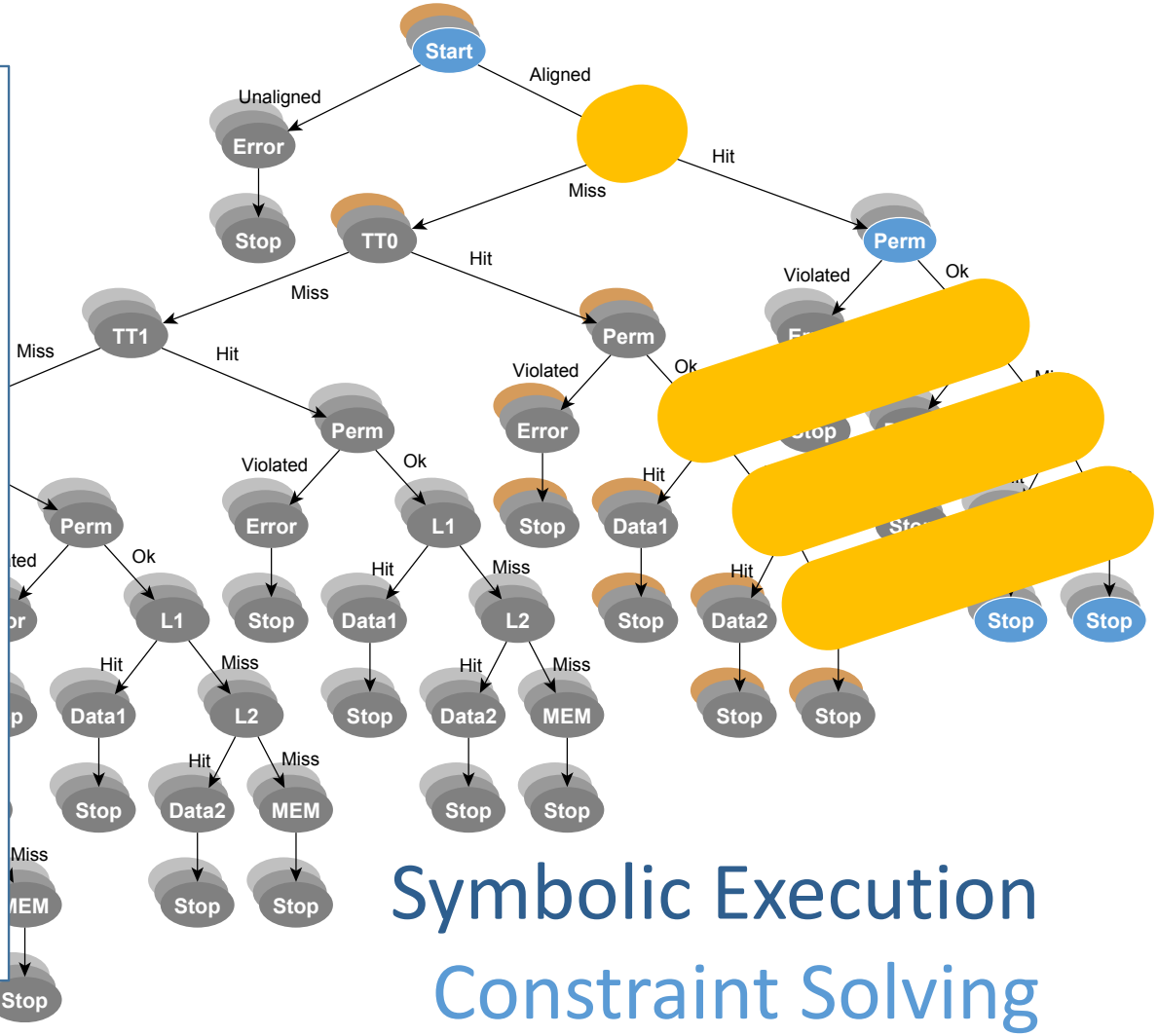
```
// Translation Table Walk
function AArch64TableWalk (...): TLBRecord {
  if ... then AArch64TranslationFault(); endif;
  if ... then AArch64AddressSizeFault(); endif;
                                  // Level 0
  c = AArch64TranslationTableWalkRepeat(c);
  if c.return_ALL != 1 then // Level 1..3
    c = AArch64TranslationTableWalkRepeat(c);
  endif;
  ...
}

// Stage 1 Address Translation
function AArch64S1Translate (...): AddrDesc {
  if S1_ENABLED then
    S1 = AArch64TranslationTableWalk(...);
    ...
  endif;
  return S1.addrdesc;
}
```

2

Ruby Шаблоны тестовых программ

```
class MmuTemplate < ArmV8BaseTemplate
  def run
    block(:engine => "memory", ...) {
      ldar reg(_), reg(_)
        do situation("access",
          miss ("TLB"), # Constraints
          hit  ("TranslationTable", 0)
        end
      stlr reg(_), reg(_)
        do situation("access",
          hit  ("TLB"), # Constraints
          miss ("L1"),
        end
    }
  end
end
```

1



Symbolic Execution

Constraint Solving

Ruby Инициализирующий Код

```
# Default Preparator for
# X Registers (REG Mode)
preparator(:target => "REG") {
  movk target, value(0,  15), 0
  movk target, value(16, 31), 1
  movk target, value(32, 47), 2
  movk target, value(48, 63), 3
}

# Optimized Preparator
preparator(:target => "REG"
  :mask => "0000XXXXXXXXXXXX") {
  movz target, value(0,  15), 0
  movk target, value(16, 31), 1
  movk target, value(32, 47), 2
}
```

1

```
# Preparator for L1 Cache
buffer_preparator(
  :target => "L1") {
  movz x0, address(0, 15), 0 # Data Address
  ...
  ldar x1, x0                # Load Data
}
```

2

```
# Preparator for Translation Table (Level 0)
buffer_preparator(
  :target => "TranslationTable"){
  mrs         x0, ttbr0_el1          # Table Base
  movz        x1, address, 0         # Entry Index
  add_sh_reg  x0, x0, x1, LSL, 3     # Entry Address
  movk        x2, entry(0, 15), 0    # Entry<0..15>
  ...
  stlr        x2, x0                 # Write
}
```

3

MicroTESK Заключение

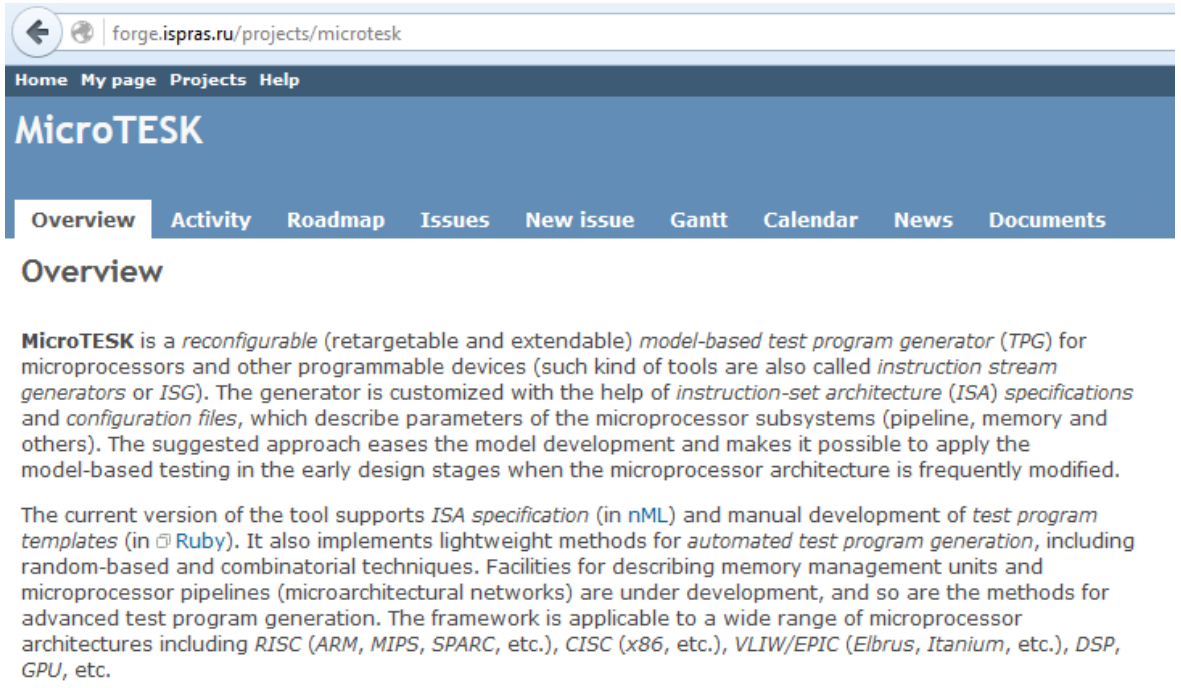
- Высокая автоматизация на базе спецификаций
- Гибкая настройка на разные архитектуры
- Расширяемость
- Поддержка ARMv8
- Open source
- Применяется в реальных проектах

Архитектура	ARMv8 AArch64
ARMv8 Reference Manual **ARM ARM Parts B, C, D, J** **Supplement 1 Parts B, E**	≈ 2600 страниц ≈ 2000 страниц ≈ 600 страниц
Поддерживаемые инструкции	≈ 750 инструкций
Спецификации ISA (nML)	≈ 11000 строк
Спецификации MMU (mmuSL)	≈ 2000 строк
Трудозатраты	≈ 1 чел./год

MicroTESK Планы на Будущее

- Online-генерация тестовых программ
- Генерация симуляторов
- Оценка тестового покрытия
- Настраиваемая компиляция
- Верификация ПО с учетом целевой архитектуры

MicroTES Контакты



http://forge.ispras.ru/projects/microtesk

microtesk-support@ispras.ru

Спасибо! Вопросы?

```
        .org 0x50000
_start: nop
        adr  x0, page_table
        msr  ttbr0_el1, x0
        movz x0, #0x8082, LSL #16
        movk x0, #0x3219, LSL #0
        msr  tcr_el1, x0
        isb  #0
        movz x0, #0x30c5, LSL #16
        movk x0, #0x0831, LSL #0
        msr  sctlr_el1, x0
        isb  #0
        movz x0, #0x0020, LSL #16
        movk x0, #0x4000, LSL #0
        ldar x1, [x0, #0]
        hlt  #0xf00d
```