

XII международная конференция  
CEE-SECR / РАЗРАБОТКА ПО

28 - 29 октября, Москва

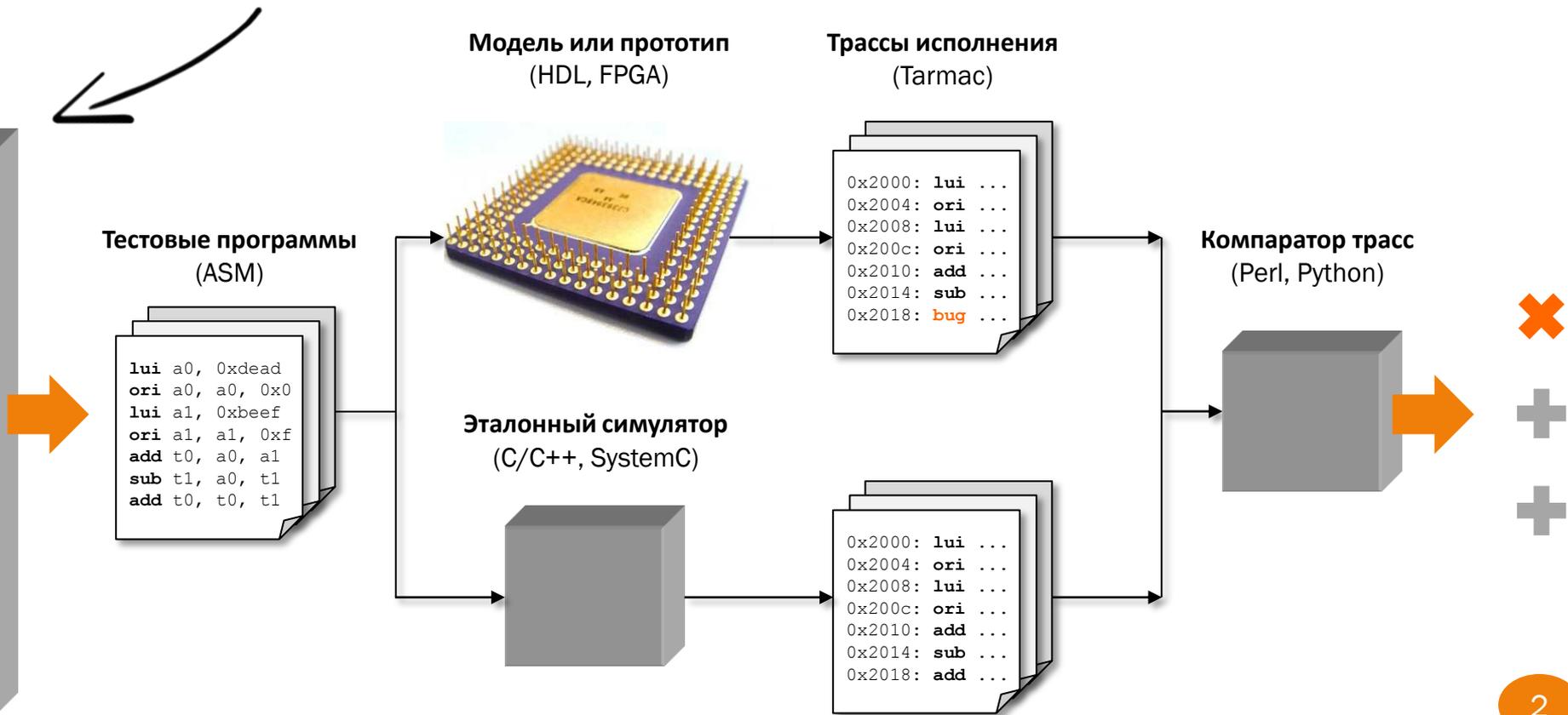


# Автоматизированная разработка генераторов тестовых программ для микропроцессоров на примере MIPS

Александр Камкин

Институт системного программирования РАН

# Это генератор тестовых программ



# Как его разработать?

**Add Word** **ADD**

31	26	25	21	20	16	15	11	10	6	5	0
SPECIAL		rs		rt		rd		0		ADD	
000000								00000		100000	
6		5		5		5		5		6	

Format: ADD rd, rs, rt MIPS32

**Operation:**

```
if NotWordValue(GPR[rs]) or NotWordValue(GPR[rt]) then
  UNPREDICTABLE
endif
temp ← (GPR[rs]31 | GPR[rs]31..0) + (GPR[rt]31 | GPR[rt]31..0)
if temp32 ≠ temp31 then
  SignalException(IntegerOverflow)
else
  GPR[rd] ← sign_extend(temp31..0)
endif
```

**MIPS**  
TECHNOLOGIES

## Руководство по системе команд

### Параметры генерации (шаблон)

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

```
TEST_PROGRAM:
addiu $sp,$sp,-24
sw $fp,16($sp)
move $fp,$sp
sw $4,24($fp)
sw $5,28($fp)
sw $0,12($fp)
j LABEL_2
nop
LABEL_3:
lw $2,12($fp)
addiu $2,$2,1
sw $2,8($fp)
j LABEL_4
nop
LABEL_5:
lw $2,12($fp)
sll $2,$2,3
move $3,$2
lw $2,24($fp)
addu $2,$3,$2
ldc1 $f2,0($2)
lw $2,8($fp)
sll $2,$2,3
move $3,$2
lw $2,24($fp)
addu $2,$3,$2
ldc1 $f0,0($2)
c.lt.d $f0,$f2
nop
bc1t LABEL_8
nop
j LABEL_6
nop
```

\$2,12(\$fp)  
\$2,\$2,3  
\$3,\$2  
\$2,24(\$fp)  
\$2,\$3,\$2  
\$f0,0(\$2)  
\$f0,0(\$fp)  
\$2,12(\$fp)  
\$2,\$2,1  
\$2,\$2,3  
\$2,24(\$fp)  
\$3,\$2  
\$2,24(\$fp)  
\$3,\$2  
\$2,24(\$fp)  
\$4,\$3,\$2  
\$2,28(\$fp)  
\$2,\$2,-1  
\$2,\$3,\$2  
\$3,\$2  
\$2,24(\$fp)  
\$2,\$3,\$2  
\$f0,0(\$2)  
\$fp,\$fp  
\$fp,16(\$sp)  
\$sp,\$sp,24  
\$31  
\$2,8(\$fp)  
\$2,\$2,1  
\$2,8(\$fp)  
\$2,8(\$fp)

\$2,8(\$fp)  
\$3,28(\$fp)  
\$2,\$2,\$3  
\$2,\$0,\$L5  
\$2,12(\$fp)  
\$2,\$2,1  
\$2,12(\$fp)  
\$3,12(\$fp)  
\$2,28(\$fp)  
\$2,\$2,-1  
\$2,\$3,\$2  
\$2,\$0,\$L3  
\$sp,\$fp  
\$fp,16(\$sp)  
\$sp,\$sp,24  
\$31

# Если только синтаксис

- Нарушение предусловий: **UNPREDICTABLE**
- Обращения к **неинициализированным** данным
- Переходы назад могут вызвать **зацикливания**
- Низкая вероятность возникновения **corner-cases**
- Невозможность создания **self-checking** тестов

# Нужно учитывать семантику



# Спецификации на nML

```
type DWORD = card(64)
```

```
reg GPR [32, DWORD]
```

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

```
  image = format("0000%5s%5s%5s0000100000", rs.image, rt.image, rd.image)
```

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

```
  action = {
```

```
    if (NotWordValue(rs) || NotWordValue(rt)) then  
      unpredictable;
```

```
  endif;
```

```
  temp = rs<31>::rs<31..0> + rs<31>::rt<31..0>;
```

```
  if (temp<32> != temp<31>) then  
    exception("IntegerOverflow");
```

```
  else
```

```
    rd = sign_extend(DWORD, temp<31..0>;
```

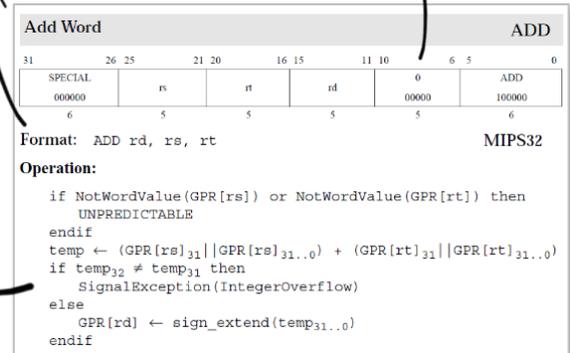
```
  endif;
```

```
}
```

Регистры

Команды

```
mode R (i: card(5)) = GPR[i]  
  syntax = format("r%d", i)  
  image = format("%5s", i)  
  action = {  
    ...  
  }  
  Режимы доступа
```



Спецификация

# Шаблоны на Ruby

Всего  $2 \times 2 = 4$  варианта

```
class MyTemplate < Template
  def initialize ... end
  def pre         ... end
  def post        ... end
  def run
    block(:combinator => 'product') {
      iterate {
        add t0, t1, t2
        do situation('Normal') end
        add t0, t1, t2
        do situation('IntegerOverflow') end
      }
      iterate {
        sub t3, t4, t5
        do situation('Normal') end
        sub t3, t4, t5
        do situation('IntegerOverflow') end
      }
    }.run
  end
end
```

Пролог / эпилог

Ассемблер

Всякие “фишки”

Инициализирующий код

```
lui r9, 0x858d
ori r9, r9, 0xc02e
lui r10, 0x58d2
ori r10, r10, 0x7219
lui r12, 0x898a
ori r12, r12, 0x49b7
lui r13, 0x0f31
ori r13, r13, 0xc65a
```

Тестовое воздействие

```
add r8, r9, r10
sub r11, r12, r13
```

# Заключение

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



Архитектура	MIPS™
Число команд	250 из 320
Объем nML спецификаций	4.5 KLOC
Трудоемкость	4 чел.-мес.

# Будущее MicroTESK

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

<http://www.isprasopen.ru>

Открытая конференция  
ИСП РАН 2016



Москва, Россия  
01-02 Декабря 2016

МЕРОПРИЯТИЯ

ПРОГРАММНЫЙ КОМИТЕТ

РЕГИСТРАЦИЯ

ДОКЛАДЫ

ПАРТНЕРЫ

МЕСТО ПРОВЕДЕНИЯ

ПРОГРАММА

ИСТОРИЯ



# MicroTESK

Verification Technology for  
Microprocessors



```
_start:      .org 0x50000
nop
add
msr x0, #0
movz x0, #0x80, LSL #16
movk x0, #0x219, 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]
```

<http://forge.ispras.ru/projects/microtesk>  
[microtesk-support@ispras.ru](mailto:microtesk-support@ispras.ru)

forge.ispras.ru/projects/microtesk

Home My page Projects Help

## MicroTESK

Overview Activity Roadmap Issues New issue Gantt Calendar News Documents

### Overview

**MicroTESK** is a *reconfigurable* (retargetable and extendable) *model-based test program generator (TPG)* for microprocessors and other programmable devices (such kind of tools are also called *instruction stream generators* or *ISG*). The generator is customized with the help of *instruction-set architecture (ISA) specifications* and *configuration files*, which describe parameters of the microprocessor subsystems (pipeline, memory and others). The suggested approach eases the model development and makes it possible to apply the model-based testing in the early design stages when the microprocessor architecture is frequently modified.

The current version of the tool supports *ISA specification* (in *nML*) and manual development of *test program templates* (in *Ruby*). It also implements lightweight methods for *automated test program generation*, including random-based and combinatorial techniques. Facilities for describing memory management units and microprocessor pipelines (microarchitectural networks) are under development, and so are the methods for advanced test program generation. The framework is applicable to a wide range of microprocessor architectures including *RISC (ARM, MIPS, SPARC, etc.)*, *CISC (x86, etc.)*, *VLIW/EPIC (Elbrus, Itanium, etc.)*, *DSP*, *GPU*, etc.

