

May 20, 2019

Holiday Inn Lesnaya,
Moscow, Russia



MicroTESK: Test Program Generation *and* Binary Code Deductive Analysis

Alexander Kamkin (kamkin@ispras.ru)



Ivannikov Institute for System Programming
Russian Academy of Sciences

- **Established in 1994** from two departments of Institute for Cybernetics Problems of the RAS
- Division of **Mathematical Sciences** of the RAS
- **200+** researchers and software engineers including approximately **15 ScDs** and **60 PhDs**
- Many employees work as professors and lecturers in leading universities: **MSU**, **MIPT**, and **HSE**
- **Microprocessor Verification Group** consists of **7+** researchers and engineers including **2 PhDs**

<http://www.ispras.ru/en>



Victor P. Ivannikov (1940 – 2016)
the Founder and the Director of
Institute for System Programming



Formalization Motivation

- **High-level synthesis (HLS)**
 - High-level HDL (Chisel, BSV, SystemC) → RTL (Verilog, VHDL)
- **Cross development tools and virtual platforms**
 - ADL (CodAL, nML) → simulator + compiler (Codusip, Imperas)
- **Functional and security verification**
 - ADL + scenarios → assembly (MicroTESK, Genesys-Pro, RAVEN)
 - HLS (Clash) + equivalence checking (Yosys, ABC, JasperGold)
 - Theorem provers and proof assistants (Isabelle/HOL, Coq, PVS)
 - ADL + SW code + properties (MicroTESK + Why3 + SMT)



Formalization Activities

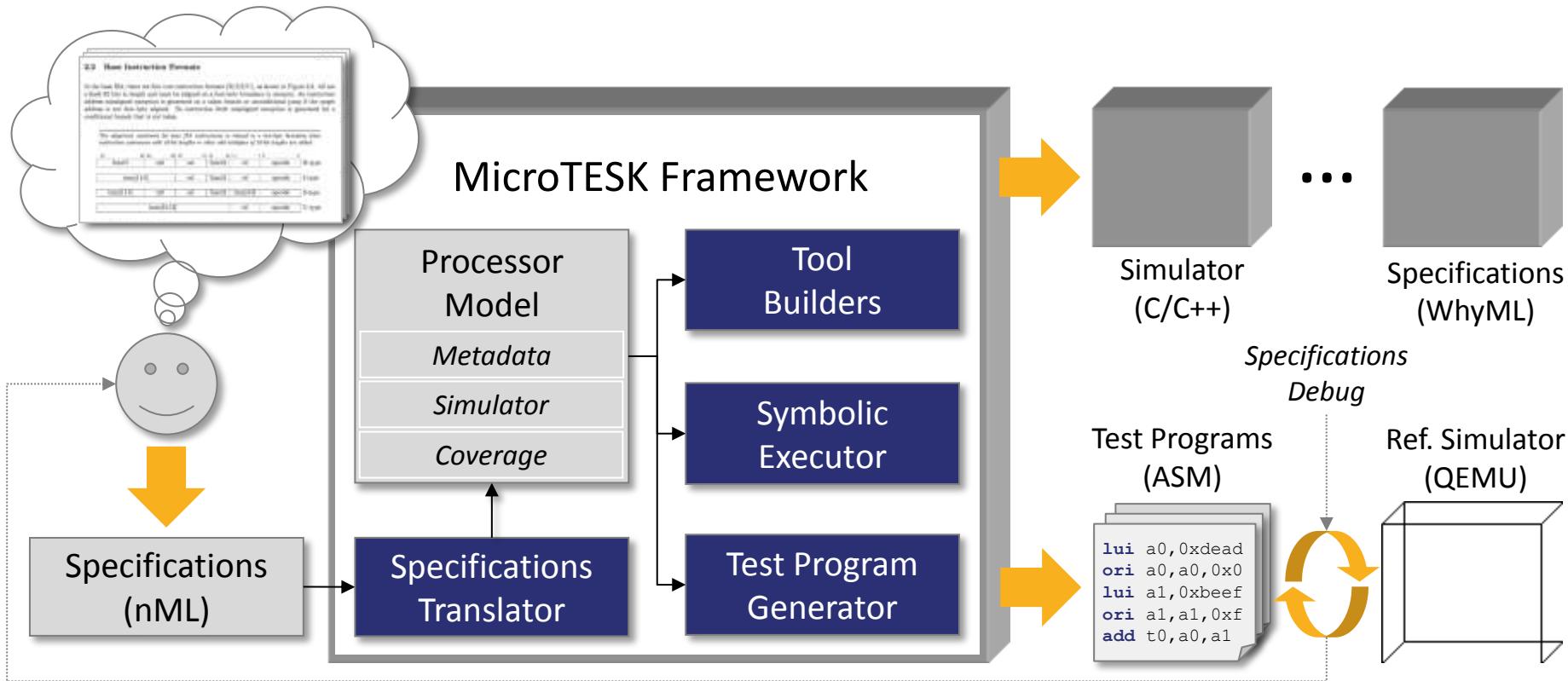
- **Technical activities**
 - SW models: C/C++ (Spike, QEMU, RV8, gem5, Imperas, etc.)
 - ISA-centric ADL: nML (ISP RAS), CodAL (Codasip)
 - High-level HDL: Chisel, Bluespec SystemVerilog
- **Research activities**
 - Functional languages: Haskell (Bluespec, Galois, MIT)
 - ISA-centric ADL: L3 DSL (SRI), nML (ISP RAS)
 - Logic frameworks: Coq (MIT), HOL4 (SRI), Why3 (ISP RAS)
- *... and possibly more (not familiar with all of them)*



Open Specifications

- <https://github.com/mit-plv/riscv-semantics>
by MIT in Haskell
- <https://github.com/cliffordwolf/riscv-formal>
by Clifford Wolf (Symbiotic EDA) in SystemVerilog
- <https://github.com/samuelgruetter/riscv-coq>
by Samuel Gruetter (MIT) in Coq
- https://github.com/rsnikhil/RISCV_ISA_Formal_Spec_in_BSV
by Rishiyur Nikhil (Bluespec) in Bluespec SystemVerilog
- <https://github.com/rsnikhil/RISCV-ISA-Spec>
by Rishiyur Nikhil (Bluespec) in Haskell
- <https://github.com/SRI-CSL/l3riscv>
by Prashant Mundkur (SRI International) in L3
- <https://forge.ispras.ru/projects/microtesk-riscv>
by ISP RAS in nML

Specification Approach and Framework



nML Language Overview

```
#ifdef RV64I  
let XLEN = 64  
...  
#else  
...  
#endif
```

```
type WORD      = card(32)  
type XWORD     = card(XLEN)  
type FLOAT32   = float(23, 8)
```

```
reg XREG [32, XWORD]  
reg PC       [XWORD]
```

```
let MEMORY_SIZE_IN_WORDS = 2 ** (XLEN - 2)  
shared mem MEM[MEMORY_SIZE_IN_WORDS, WORD]
```

```
op add (rd: X, rs1: X, rs2: X)  
syntax = format("add %s, %s, %s",  
    rd.syntax, rs1.syntax, rs2.syntax)  
image  = format("0000000%s%s000%s0110011",  
    rs2.image, rs1.image, rd.image)  
action = {  
    rd = rs1 + rs2;  
}
```

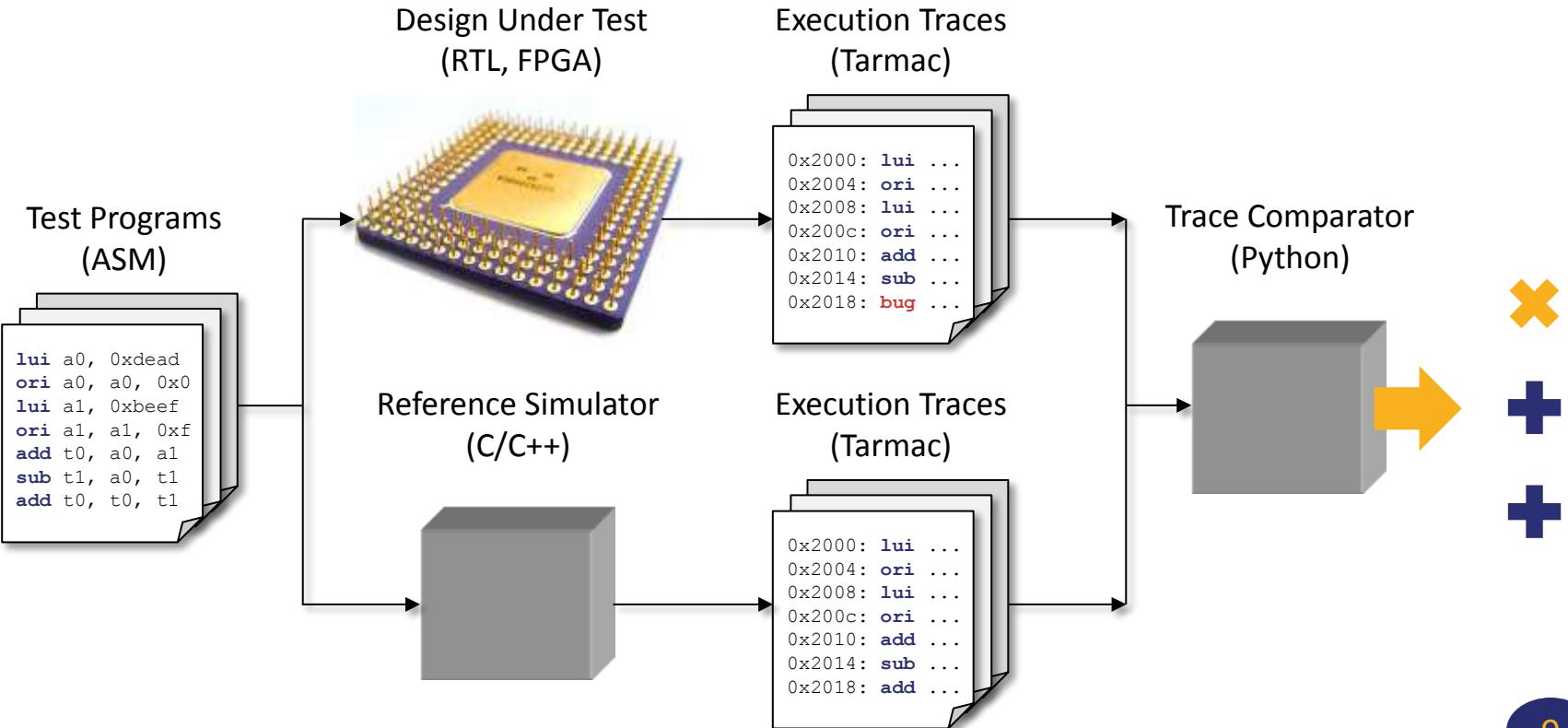
```
mode X (i: card(5)) = XREG[i]  
syntax = format("%s",  
    if i==0 then ZERO().syntax  
    elif i==1 then RA().syntax  
    elif i==2 then SP().syntax  
    elif i==3 then GP().syntax  
    elif i==4 then TP().syntax  
    elif i>=5 && i<=7 then  
        Temp(coerce(card(3), i-5)).syntax  
    elif i>=8 && i<=9 then  
        Saved(coerce(card(4), i-8)).syntax  
    elif i>=10 && i<=17 then  
        Func(coerce(card(3), i-10)).syntax  
    elif i>=18 && i<=27 then  
        Saved(coerce(card(4), i-16)).syntax  
    else  
        Temp(coerce(card(3), i-25)).syntax  
    image = format("%5s", i)
```



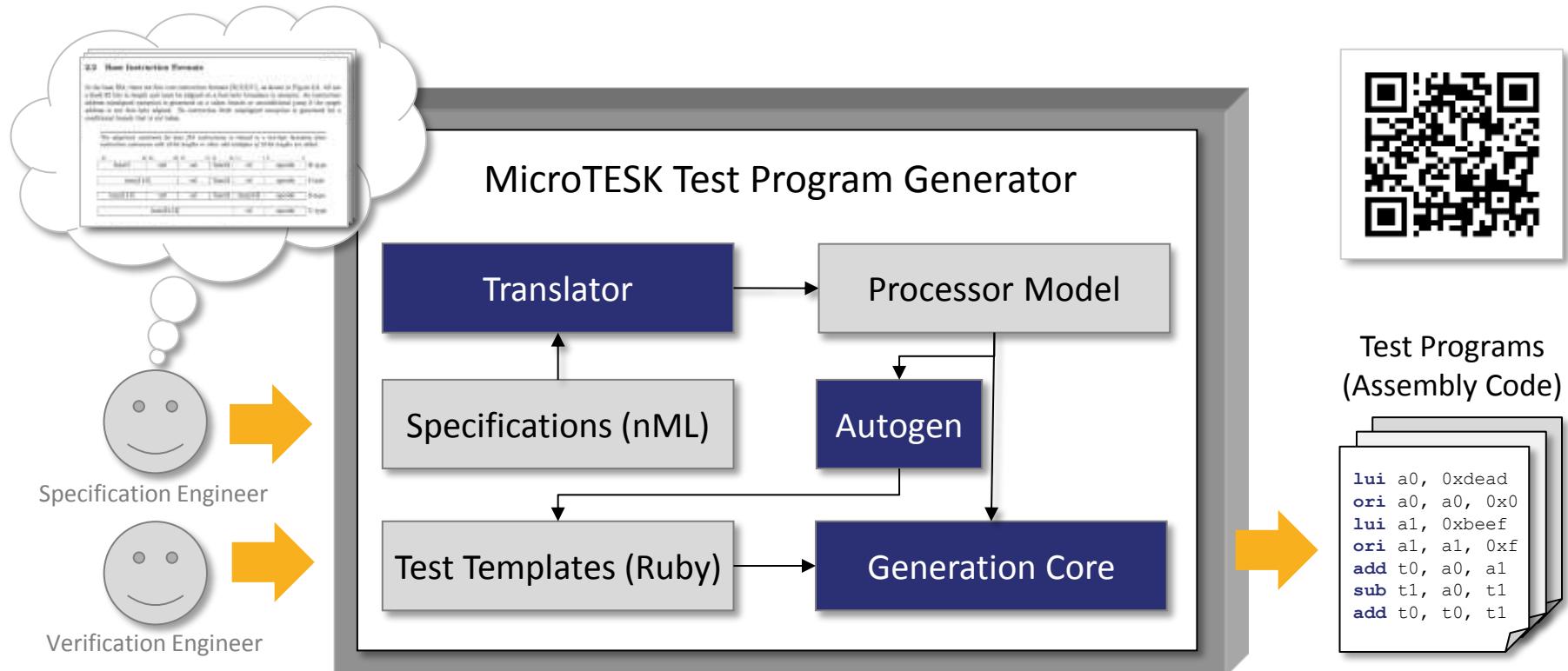
RISC-V Specifications

Specifications Characteristic					Value				
RISC-V Instruction Set Manual Vol. I: User-Level ISA (v. 2.2)					145 pages (94 pages in chapters 2-18)				
Base	Extension				Base	Extension	In Total		
RV32I	M	F	C	P	24 pages	44 pages	68 pages		
RV32E	A	D	B	V					
RV64I		Q	J	N	2 pages	24 pages	26 pages		
RV128I		L	T						
# Specified Instructions					200+ instructions				
ISA Specifications Size (nML)					4 500 LOC (w/o comments)				
MMU Specification Size (MMUSL)					0 LOC (unspecified)				

Application 1: Test Program Generation (1/3)



Application 1: Test Program Generation (2/3)



Application 1: Test Program Generation (3/3)

```
class MyTemplate < RiscVBaseTemplate
  def run
    block(:combinator => 'product',
          :compositor => 'random') {
      iterate {
        xor x(_), x(_), x(_) do situation(...) end
        lui x(_), -
      }
      iterate {
        and x(_), x(_), x(_)
        or x(_), x(_), x(_)
      }
    }.run
  end
end
```

- *Combinatorial Brute Force*
- *Randomization*
- *Constraint Solving*
- *Self Checking*



```
# Single Test Case
# Initialization
ori a7, a7, 0x2d7
slli a7, a7, 0xb
ori a7, a7, 0x1
slli a7, a7, 0xb
ori a7, a7, 0x3d2
ori t3, t3, 0x164
slli t3, t3, 0xb
ori t3, t3, 0x52b
slli t3, t3, 0xb
ori t3, t3, 0x24e
# Stimulus
and s4, a7, a7
xor s8, s4, t3
```

- Test templates similar to **RISC-V Foundation's Tests**
<https://github.com/riscv/riscv-tests>
- Test templates similar to **RISC-V Torture Test Generator**
<https://github.com/ucb-bar/riscv-torture>

$2 \times 2 = 4$ test cases

Code is distributed under
Apache License, v2.0

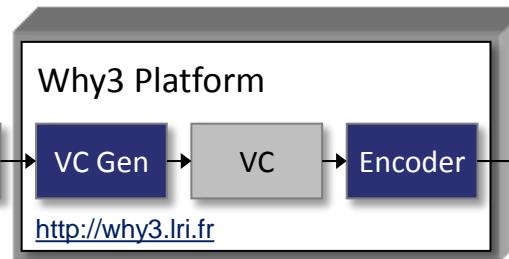
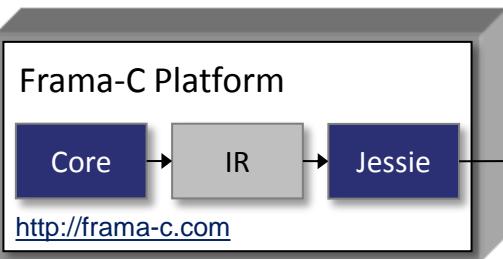


Application 2: Binary Code Verification (1/2)

Source Code

(C + ACSL)

```
//@ requires  
//@ ensures  
int idiv(...) {  
//@ loop inv  
while (...) {...}  
}
```



Provers
(Coq, CVC4)

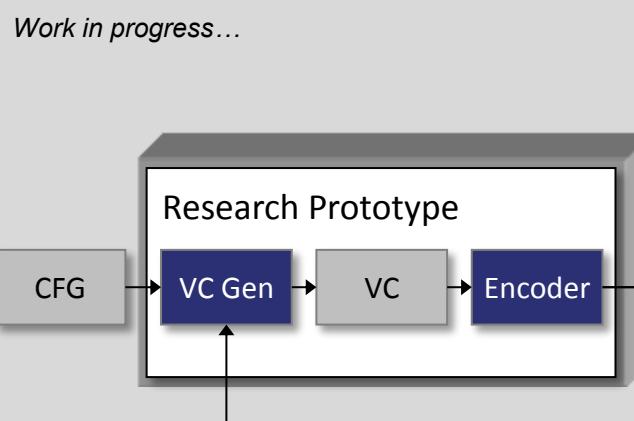
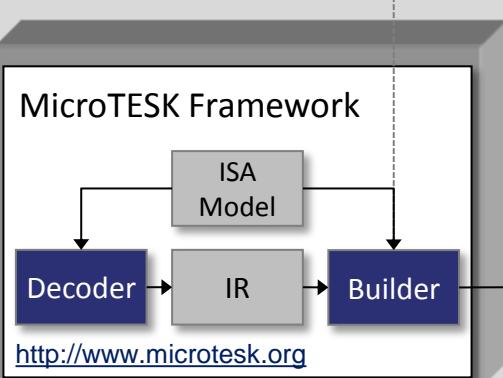
...

X



Binary Code

```
006f 04c0 2f73  
3420 0f93 0080  
0a63 03ff 0f93  
0090 0663 03ff  
0f93 00b0 0263  
03ff 0f13 ...
```



Annotations
(SMT-LIB)

Work in progress...

Application 2: Binary Code Verification (2/2)

```
//@ requires a >= 0;
//@ requires b > 0;
//@ ensures a == b*\result + *r;
//@ ensures 0 <= *r < b;
int idiv (int a, int b, int *r) {
    int q = 0; *r = a;
    //@ loop invariant (a == b*q + *r);
    //@ loop invariant (0 <= *r)
    while (*r >= b) {
        q++; *r -= b;
    }
    //@ assert (a == q*b + *r);
    //@ assert (0 <= *r < b);
    return q;
}
```

Source Code
(C + ACSL)

Binary Code
(CFG)

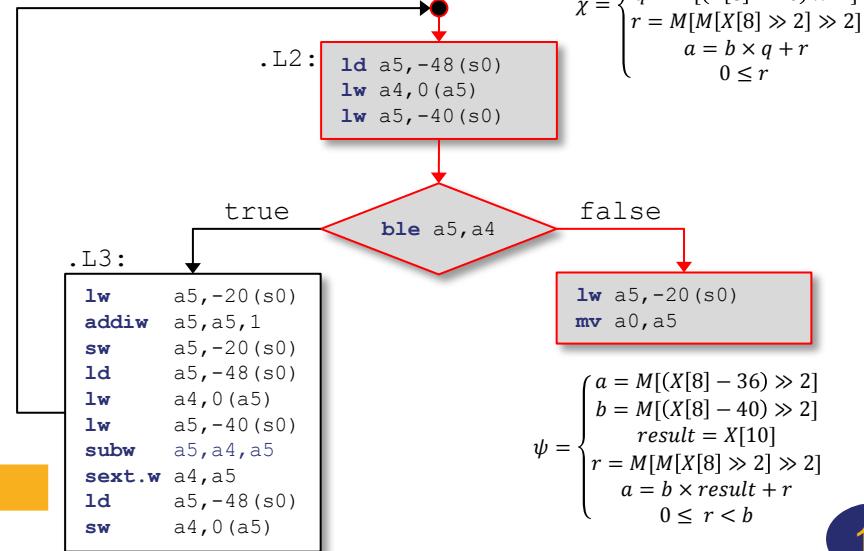
idiv:

```
mv a5,a0
mv a4,a1
sd a2,-48(s0)
sw a5,-36(s0)
mv a5,a4
sw a5,-40(s0)
sw zero,-20(s0)
ld a5,-48(s0)
lw a4,-36(s0)
sw a4,0(a5)
```

$$\varphi = \begin{cases} a = X[10] \\ b = X[11] \\ a \geq 0 \\ b > 0 \end{cases}$$

Verification Conditions
(SMT-LIB)

```
(assert (= op0.rd.i!1 #b01111))
(assert (= op0.rsl.i!1 #b01000))
(assert (= op0.imm!1 #b111111010000)) ; ld a5, -48(s0)
(assert (and
  (= op0.rsl!1 (select X!1 op0.rsl.i!1))
  (= addr!1 (bvadd op0.rsl!1 ((_ sign_extend 52) op0.imm!1)))
  (= index!1 (bvlshr addr!1 #b0...010)))
...
(= dword!9 (select M!1 index!1))
(= X!2 (store X!1 op2.rd.i!1 dword!9))
...
(assert (and ; mv a0, a5
  (= X!10 (store X!9 op5.rd.i!1 op5.rd!1))
  (= op5.rsl!1 (select X!9 op5.rsl.i!1))
  (= op5.rd!1 (bvadd op5.rsl!1 ((_ sign_extend 52) op5.imm!1))))
```



Future Work Directions

- **RISC-V specifications**
 - Specification of subsets (RV128I, Q, L, B, T, J, P, V, and N)
 - Specification of MMU (address translation mechanisms)
- **MicroTESK framework**
 - Online test program generation (post-silicon verification)
 - RTL generation (golden model for equivalence checking)
- **Specifications validation**
 - Testing (self checking, co-simulation, coverage analysis)
 - Formal equivalence checking (HLS to Verilog + BMC)

ISP RAS OPEN: December 5-6, 2019



MicroTESK

Verification Technology for Microprocessors



<http://www.microtesk.org>

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.

micromesek-support@ispras.ru



Thank

inoy