Generating Fast Functional Simulators from Formal Specifications

2023 UK Systems Research Challenges Workshop

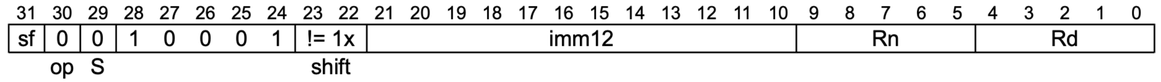
Ferdia McKeogh, Dr. Tom Spink, Prof. Al Dearle

fm208@st-andrews.ac.uk

Instruction Set Architectures (ISAs)

Instructions

Syntax




```
add x4, x3, #0x5
```

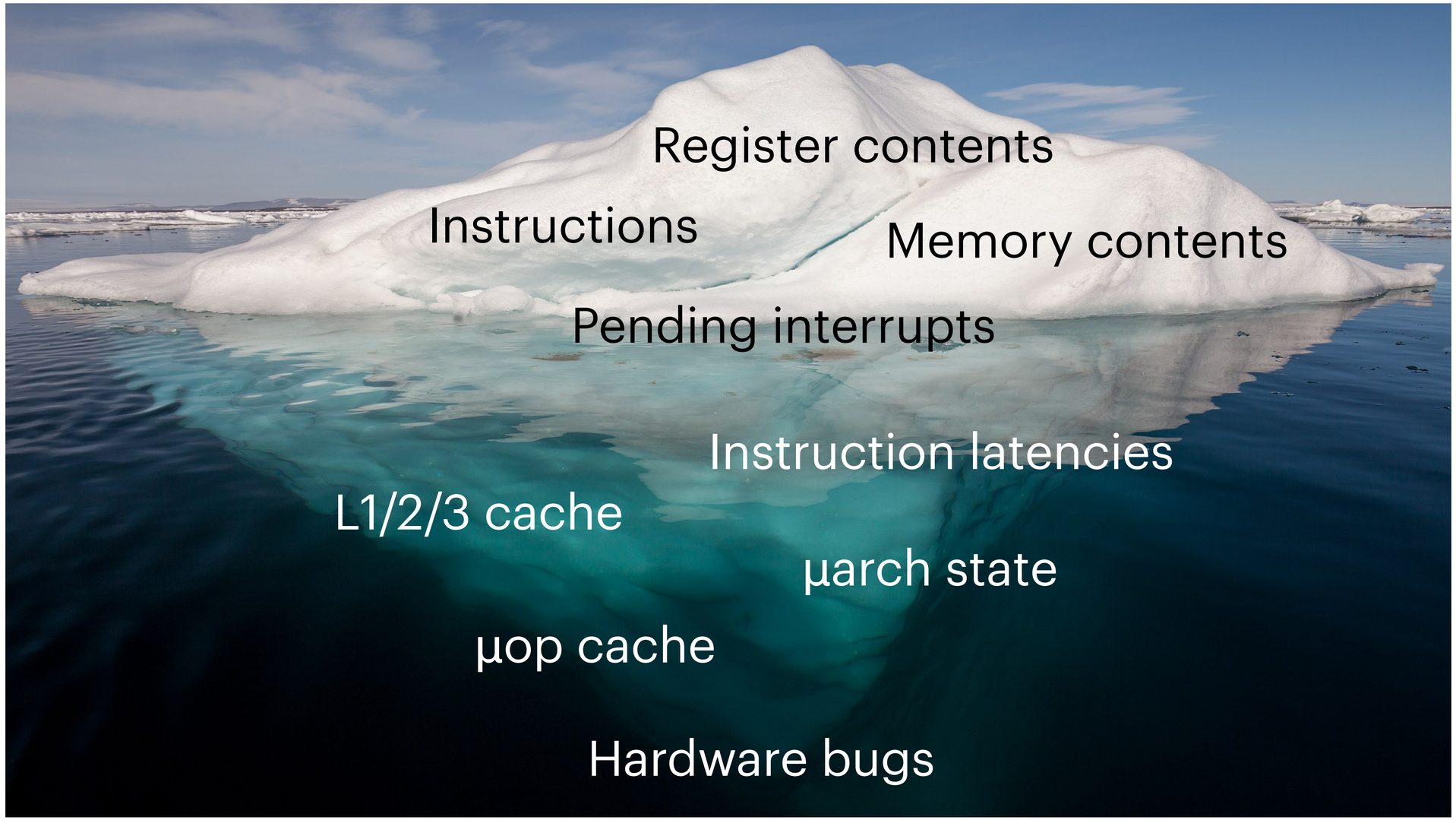
```
91 00 14 64
```

```
1 0 0 10001 0 000000000101 00011 00100
```

Semantics

```
Rd = Rn + imm12
```

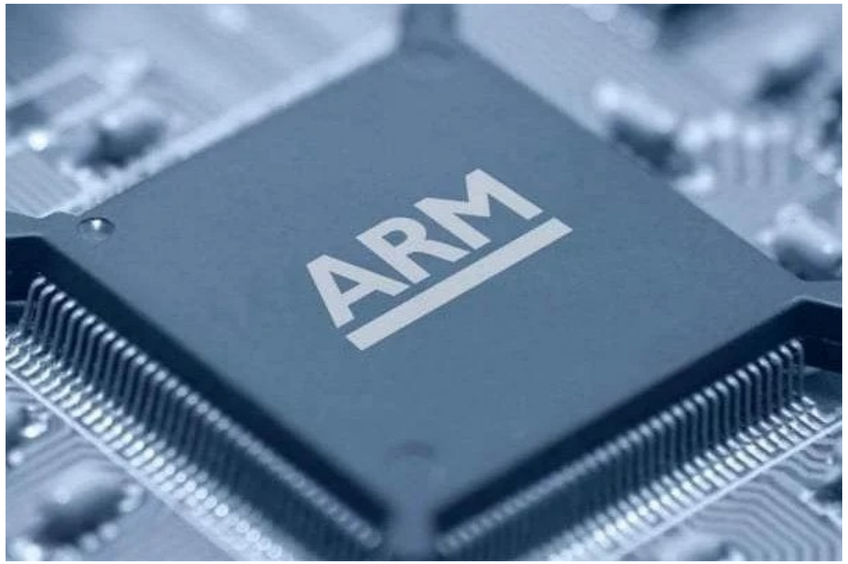
The ISA-berg



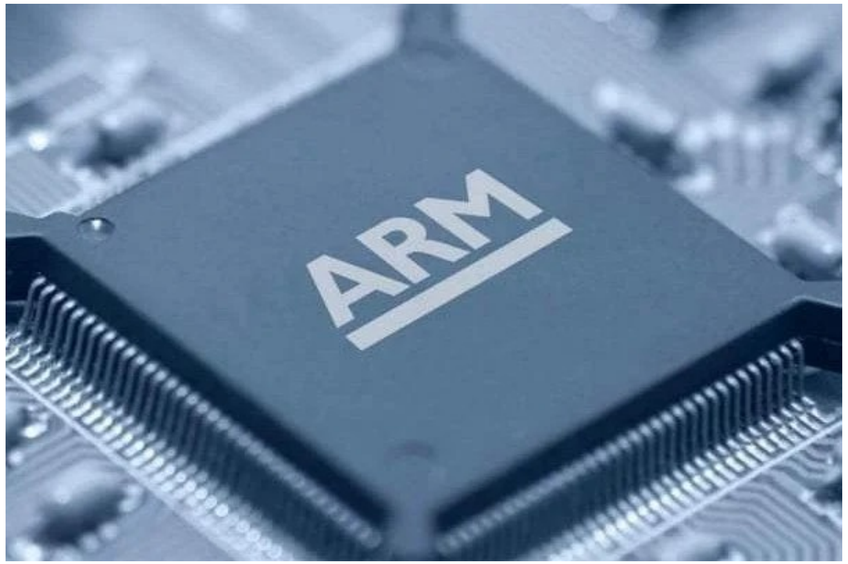
Andreas Weith, 2015

Amazon UK, 2022

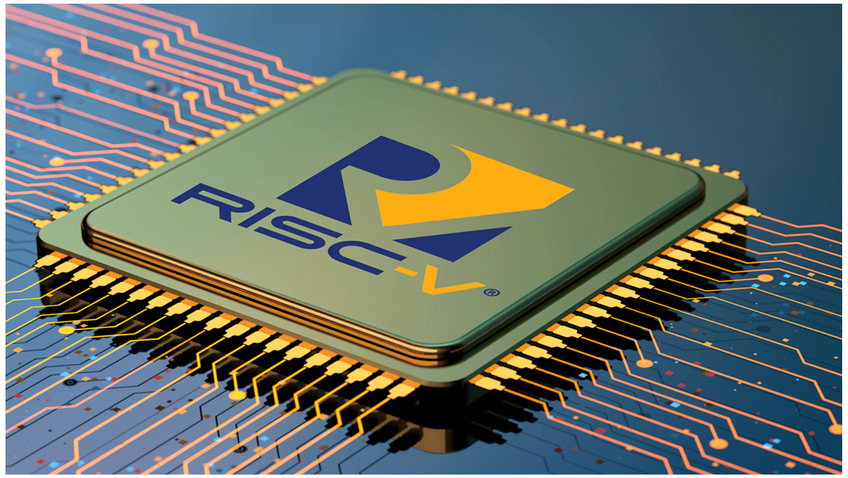
Amazon UK, 2022



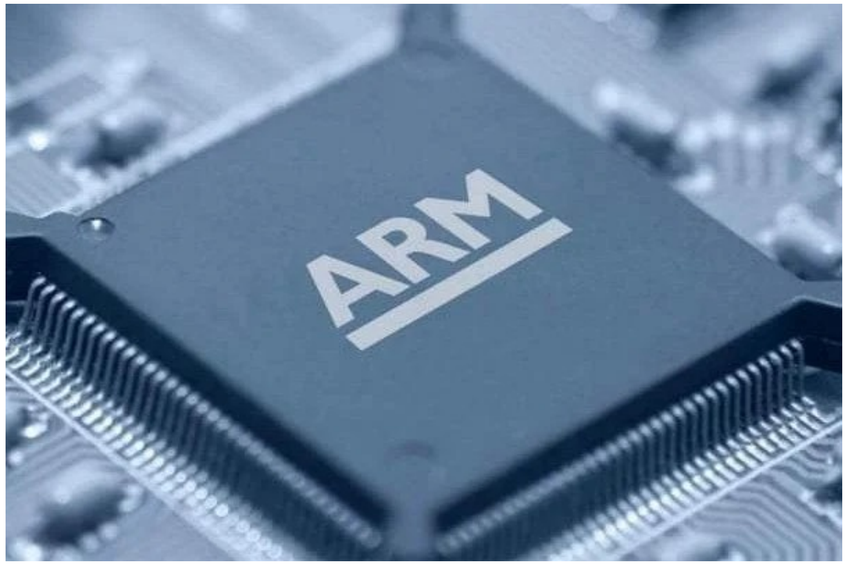
Trusted Reviews, 2022



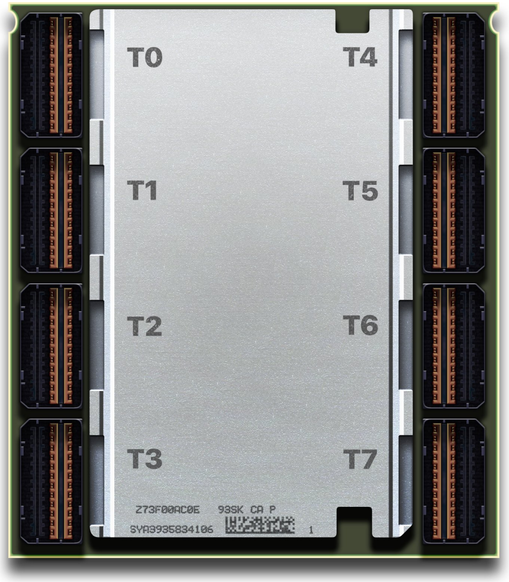
Trusted Reviews, 2022



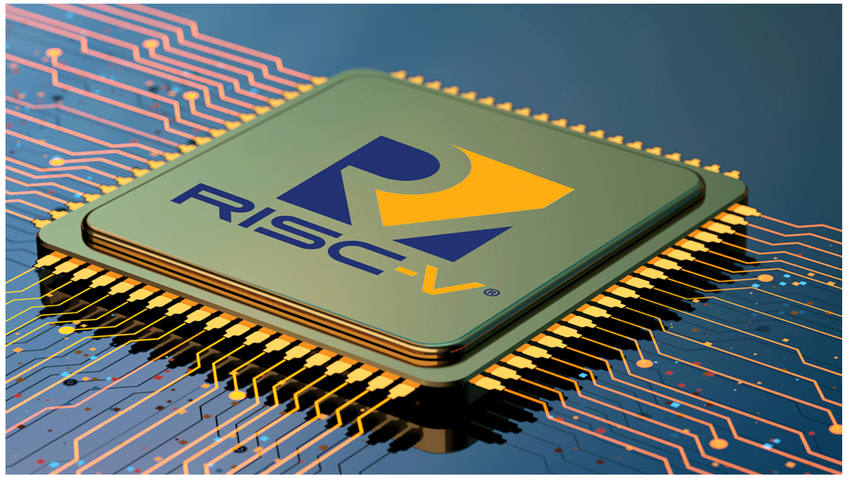
Siemens PLM, 2022



Trusted Reviews, 2022



Wikipedia user “henkriok”, 2021



Siemens PLM, 2022

Problems

Problems

- Legacy software compiled for old architectures

Problems

- Legacy software compiled for old architectures
- Development machine with different ISA to target machine

Problems

- Legacy software compiled for old architectures
- Development machine with different ISA to target machine
- Need to develop software for future architectures

Solution?

Solution?
ISA Simulation

Solution?
ISA Simulation

- Legacy architectures that are no longer supported?
- Need to run and test software on development machine?
- Want to develop software for future architectures?

Solution?
ISA Simulation

- Legacy architectures that are no longer supported? Simulate!
- Need to run and test software on development machine?
- Want to develop software for future architectures?

Solution?
ISA Simulation

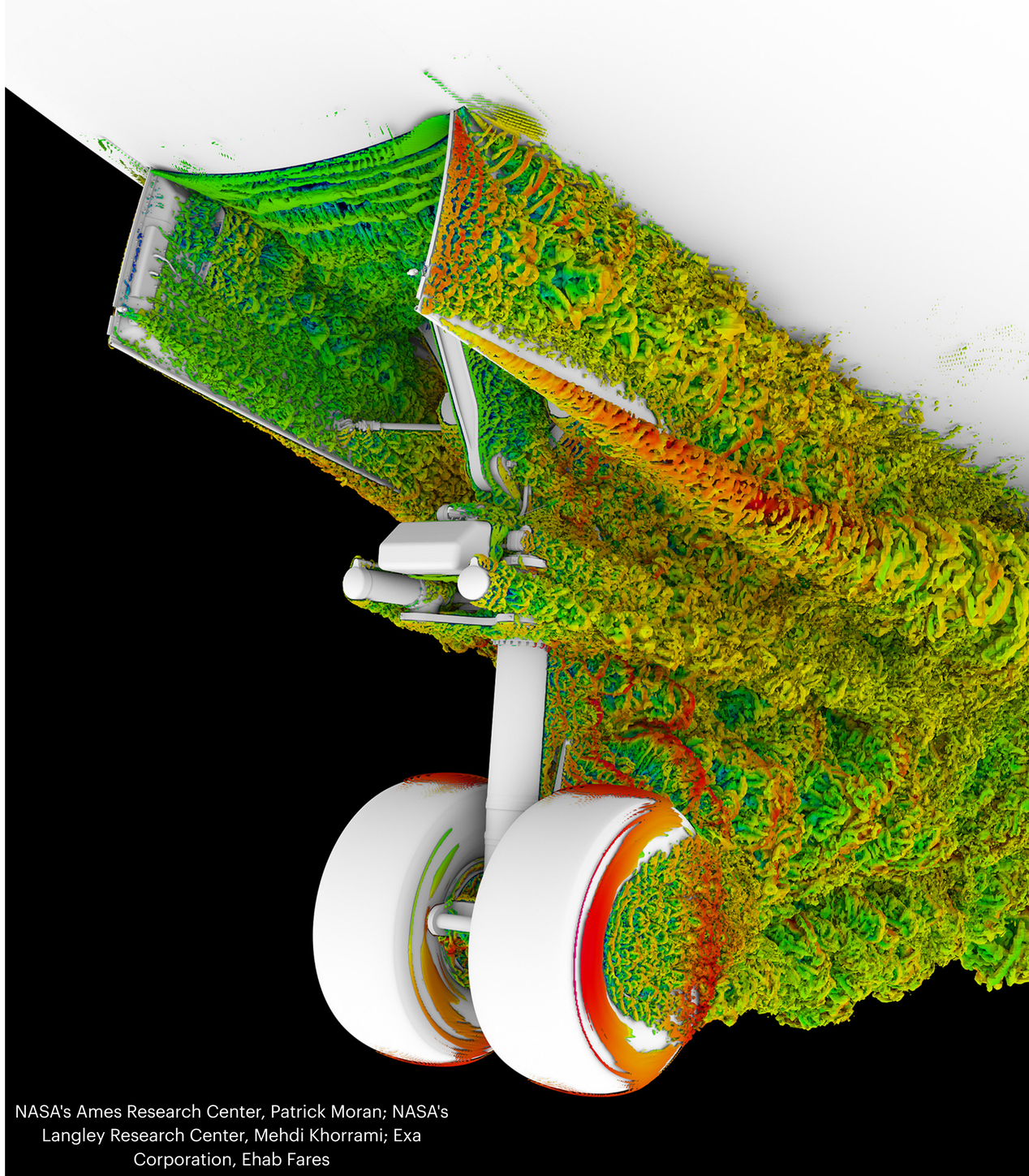
- Legacy architectures that are no longer supported? Simulate!
- Need to run and test software on development machine? Simulate!
- Want to develop software for future architectures?

Solution?
ISA Simulation

- Legacy architectures that are no longer supported? Simulate!
- Need to run and test software on development machine? Simulate!
- Want to develop software for future architectures? Simulate!

Simulation

Modelling of a physical object for inspection or manipulation



NASA's Ames Research Center, Patrick Moran; NASA's Langley Research Center, Mehdi Khorrami; Exa Corporation, Ehab Fares

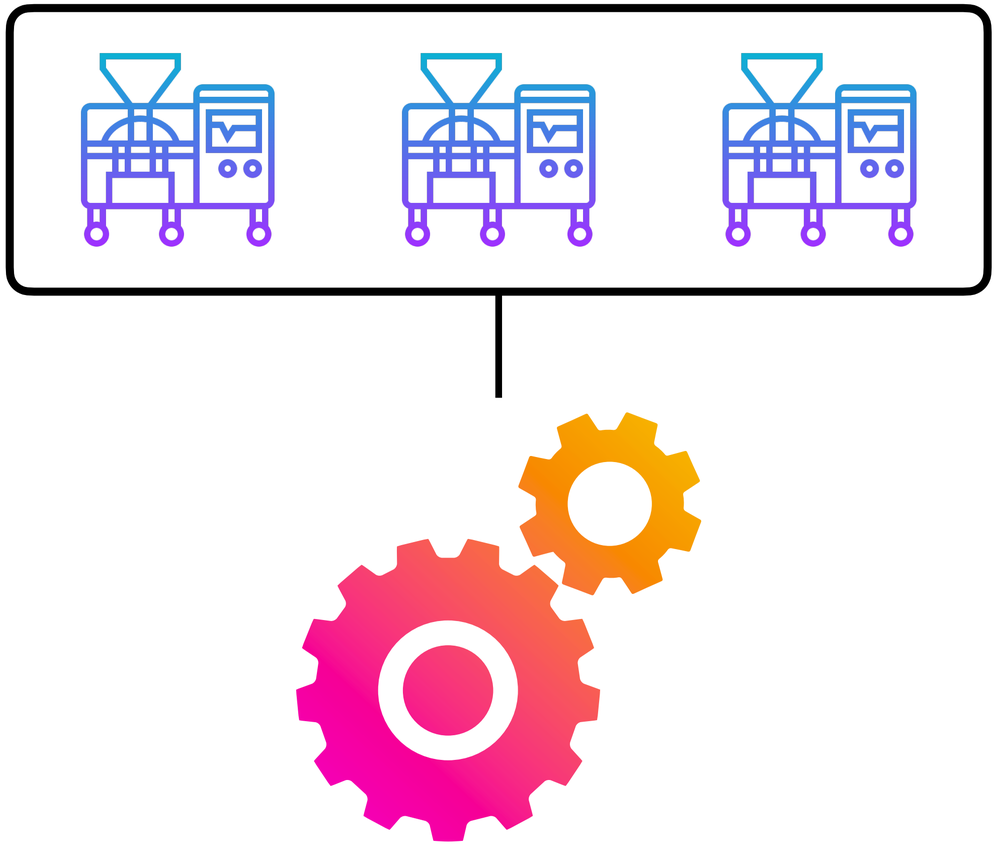
Virtualisation

Virtual implementation of a physical object

creativemarket.com/eucalyp

Emulation

Techniques to provide virtual implementations of physical objects



Sail

"Language and compiler for describing the ISA semantics of processors"

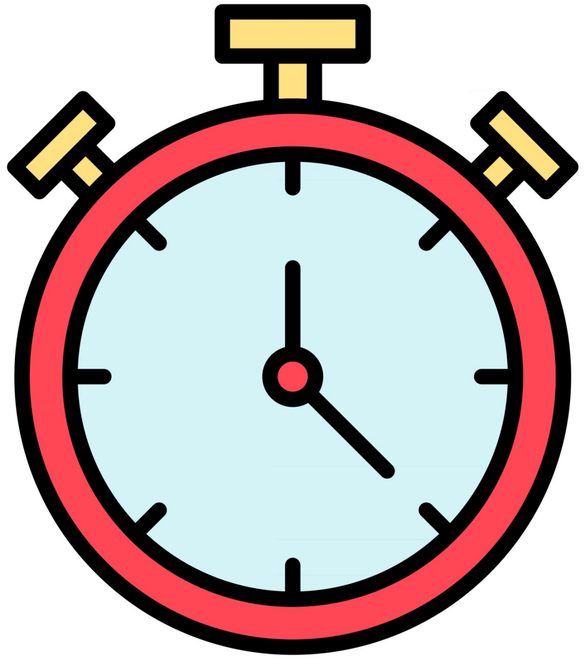
- Developed by Cambridge for formal verification
- General purpose but designed for ISA specification
- Many architecture descriptions already implemented and verified
- Some even use it as their authoritative specification

```
register VBAR_EL1 : bits(64)

val get_VBAR_NS : unit -> bits(32) effect {rreg, undef}

function get_VBAR_NS () = {
    r : bits(32) = undefined : bits(32);
    let r = __SetSlice_bits(32, 32, r, 0, slice(VBAR_EL1, 0, 32));
    r
}
```

Sail

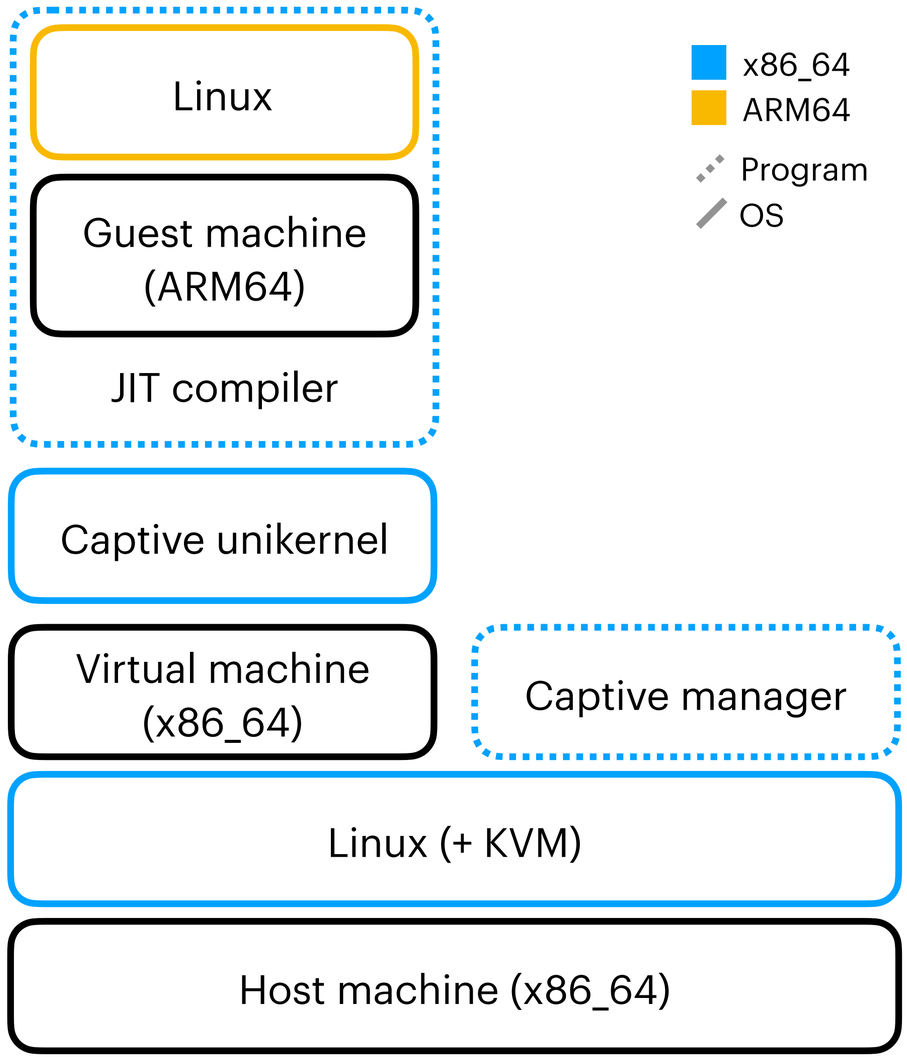
“Language for describing the ISA semantics of processors”

- Slow
  - No hardware acceleration
  - No JIT
  - No decode/execute separation



Captive

- Places JIT compiler inside VM for greater control over hardware
- Accelerated address translation
- >2x faster than QEMU
- Architecture specification used to create JIT compiler for performing DBT



GenC

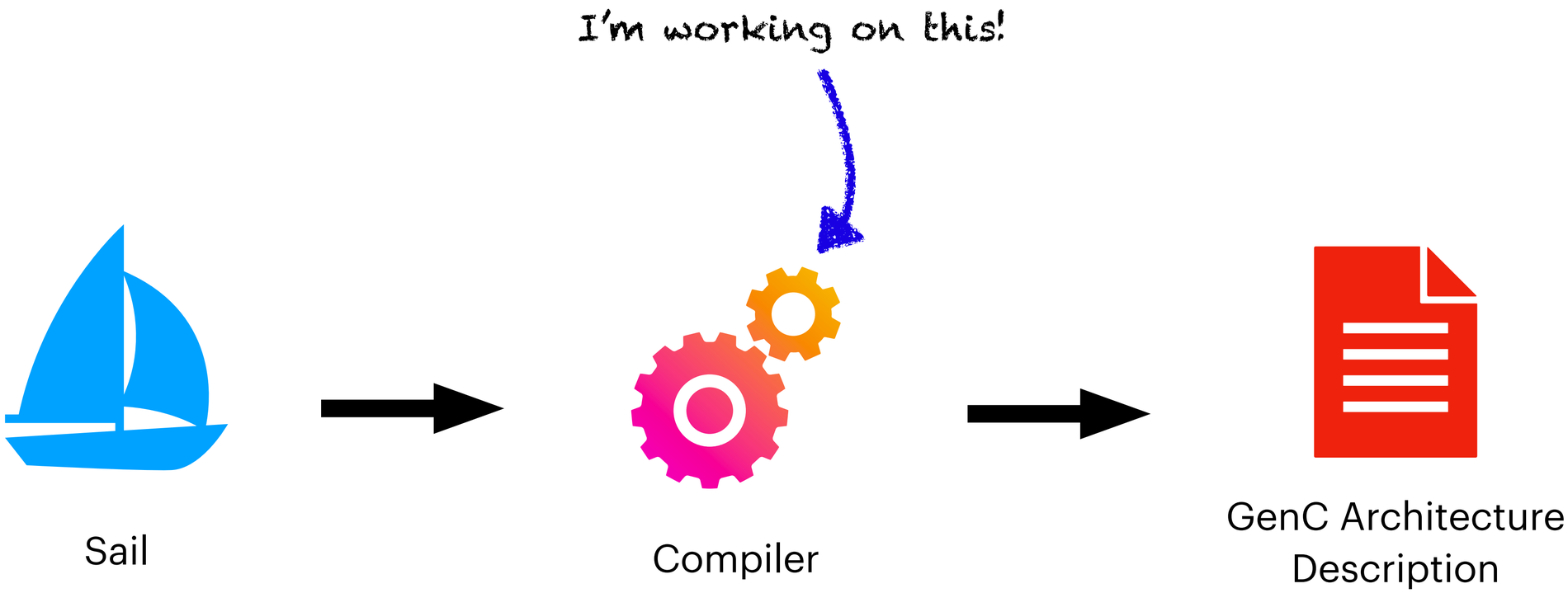
- Domain specific language for describing architectures
- Declarative style
- C-like instruction behaviour implementation
- Manual transcription ☹

```
/*
RISC-V
*/
AC_ARCH(riscv32)
{

        ac_mem Mem(4, 4, little, 1);

        // General Purpose Registers
        ac_regspace(128) {
                // bank NAME (TYPE, OFFSET, COUNT, REG-STRIDE, # ELEMS, ELEM-SIZE, ELEM-STRIDE)

                bank GPR (uint32, 0, 32, 4, 1, 4, 4);
                slot SP (uint32, 4, 8) SP; // TODO: find out the real offset of the stack pointer
        }

        ac_regspace(4) {
                slot PC (uint32, 4, 0) PC;
        }
```

```
19  // 32-Base Instructions
20  execute(addi)
21  {
22          sint32 imm = inst.imm;
23          imm <<= 20;
24          imm >>= 20;
25
26
27          typename word_t rs = read_gpr(inst.rs1);
28
29          rs += (typename sword_t)imm;
30
31          write_register_bank(GPR, inst.rd, rs);
32
33  }
34
```

Sail

GenC Architecture Description


I'm working on this!
Sail
Compiler
GenC Architecture Description

Why is this hard?

```
function integer_arithmetic_addsub_immediate (d, datasize, imm, n, setflags, sub_op) = {
    result : bits('datasize) = undefined : bits('datasize);
    let operand1 : bits('datasize) = if n == 31 then SP() else X(n);
    operand2 : bits('datasize) = imm;
    nzcv : bits(4) = undefined : bits(4);
    carry_in : bits(1) = undefined : bits(1);
    if sub_op then {
        operand2 = ~(operand2);
        carry_in = 0b1
    } else {
        carry_in = 0b0
    };
    (result, nzcv) = AddWithCarry(operand1, operand2, carry_in);
    if setflags then {
        (PSTATE.N @ PSTATE.Z @ PSTATE.C @ PSTATE.V) = nzcv
    };
    if d == 31 & ~(setflags) then {
        SP() = result
    } else {
        X(d) = result
    }
}
```

Sail

```
execute(addi)
{
        uint64 imm = decode_imm(inst.imm12, inst.shift);
        uint64 op1 = read_gpr_sp(inst.sf, inst.rn);

        if (inst.S) {
                if (inst.sf == 0) {
                        write_gpr32(inst.rd, __builtin_adc32_flags((uint32)op1, (uint32)imm, 0));
                } else {
                        write_gpr64(inst.rd, __builtin_adc64_flags(op1, imm, 0));
                }
        } else {
                write_gpr_sp(inst.sf, inst.rd, op1 + imm);
        }
}
```

GenC

Why is this hard?

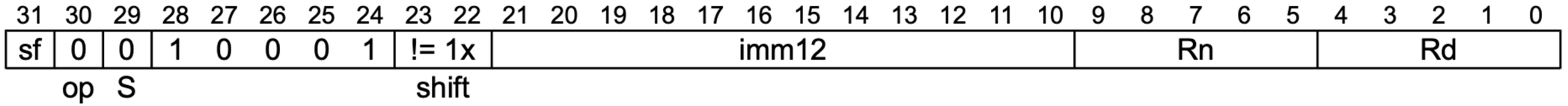
Sail

- General purpose
- No required structure
- Decoding and execution not separate
- Polymorphism, powerful type system
- No unstructured control flow*

GenC

- Declarative
- Structured
- Distinct instruction decode and execution
- No unstructured control flow

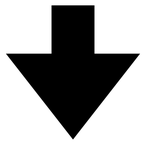
Instruction Decode




```
ac_format ADD_SUB_IMM = "%sf:1 %op:1 %S:1 0x11:5 %shift:2 %imm12:12 %rn:5 %rd:5";
```

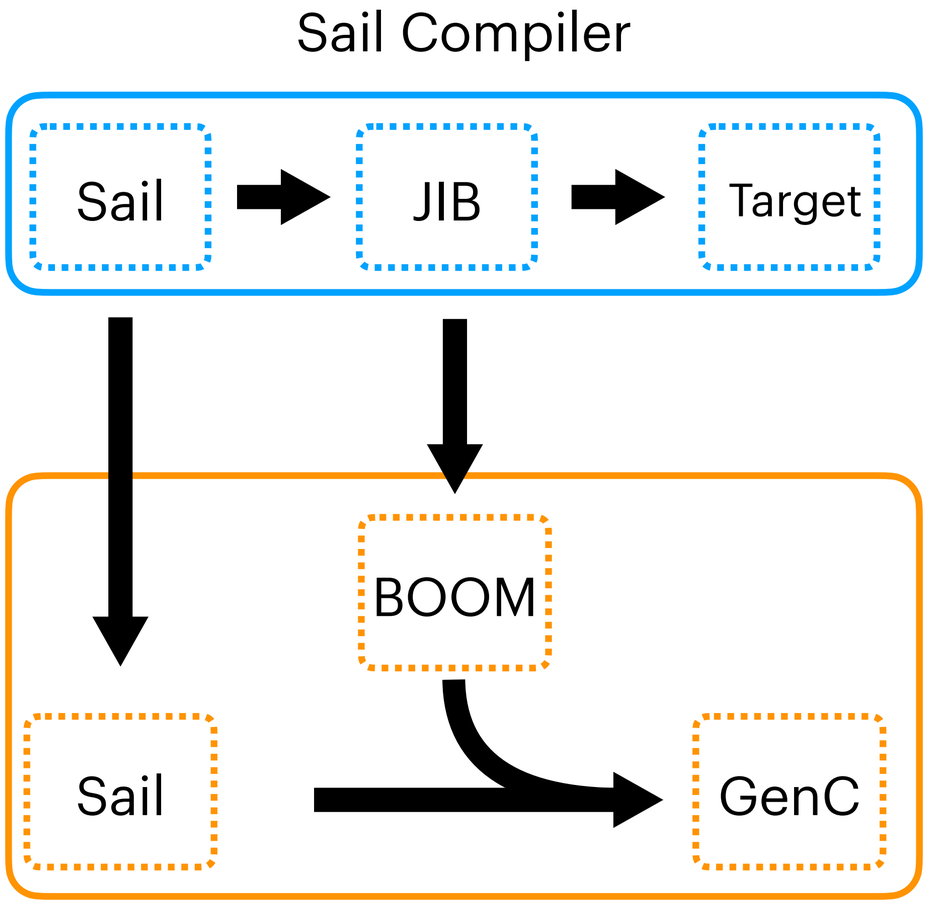
Instruction Decode

```
function clause decode64 ((_ : bits(1) @ 0b0010001 @ _ : bits(24) as op_code) if SEE < 1066) = {
    SEE = 1066;
    Rd : bits(5) = op_code[4 .. 0];
    Rn : bits(5) = op_code[9 .. 5];
    imm12 : bits(12) = op_code[21 .. 10];
    shift : bits(2) = op_code[23 .. 22];
    S : bits(1) = [op_code[29]];
    op : bits(1) = [op_code[30]];
    sf : bits(1) = [op_code[31]];
    integer_arithmetic_addsub_immediate_decode(Rd, Rn, imm12, shift, S, op, sf)
}
```

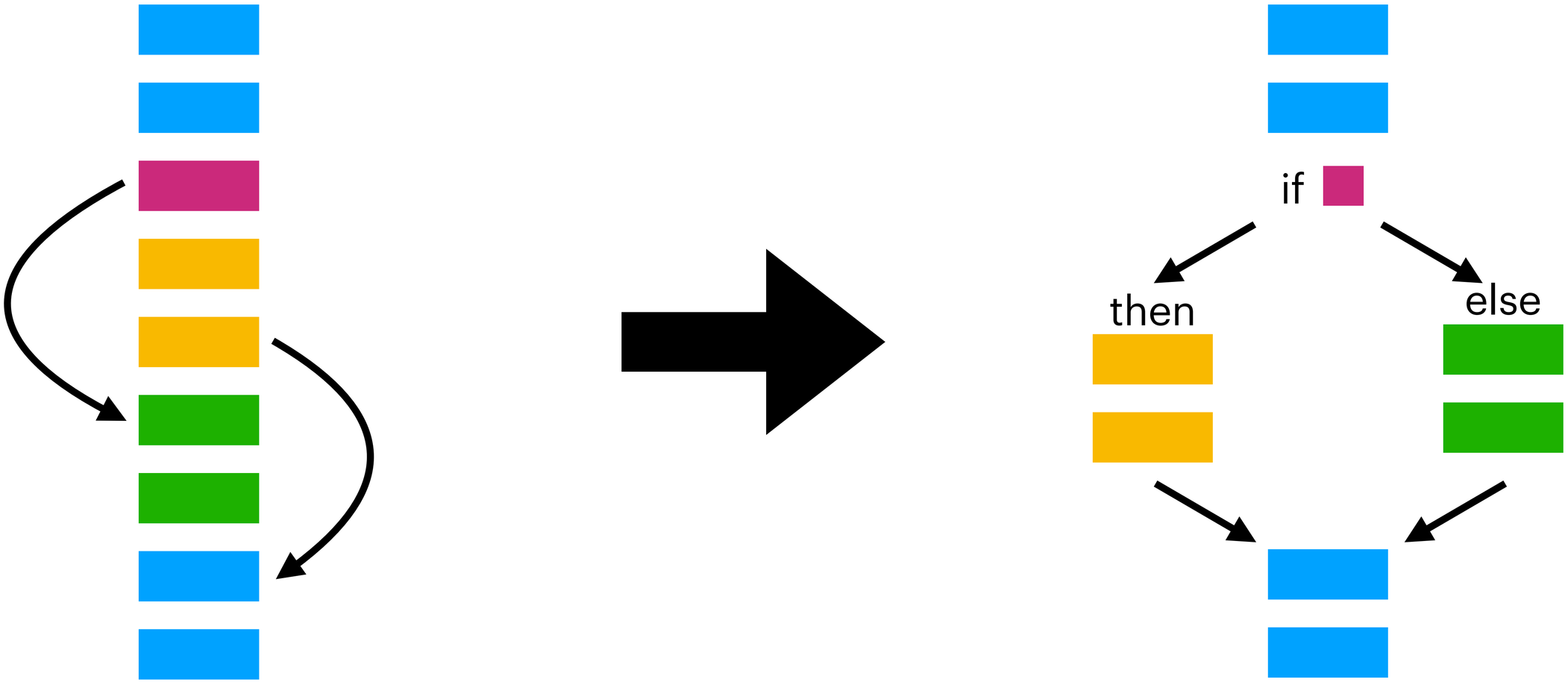


```
ac_format ADD_SUB_IMM = "%sf:1 %op:1 %S:1 0x11:5 %shift:2 %imm12:12 %rn:5 %rd:5";
```

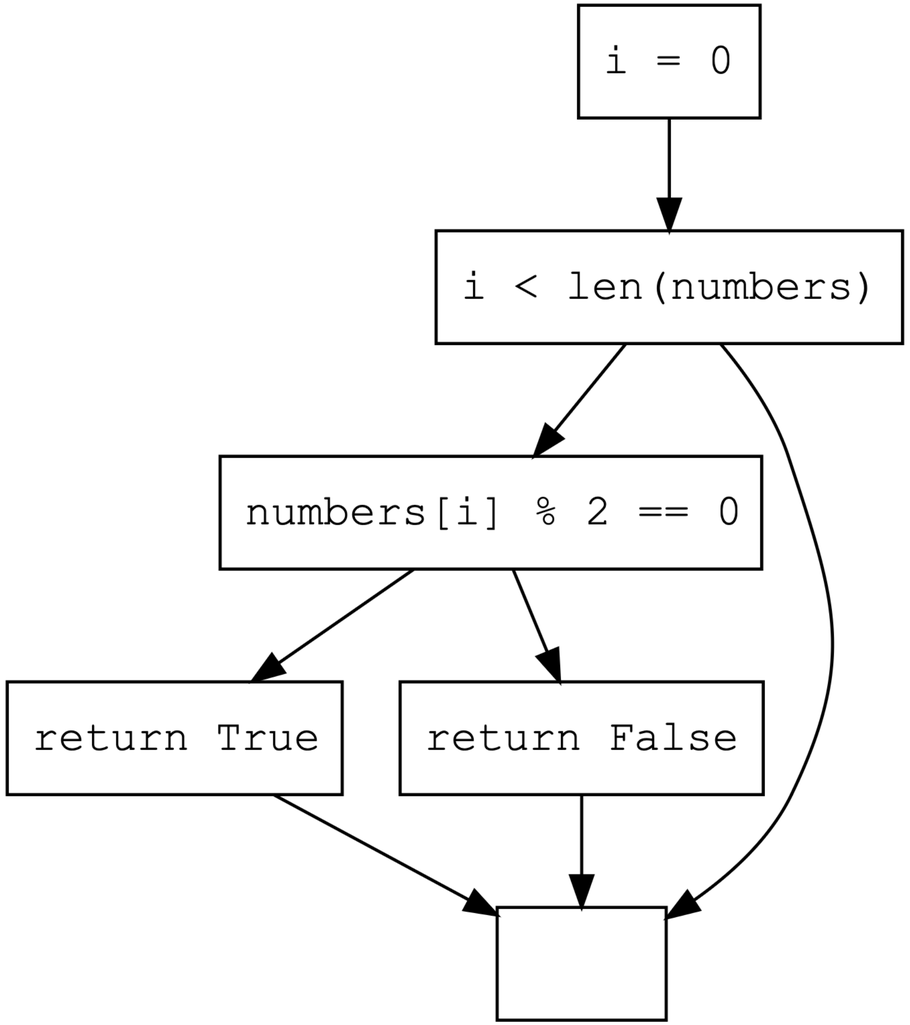
Instruction Execute



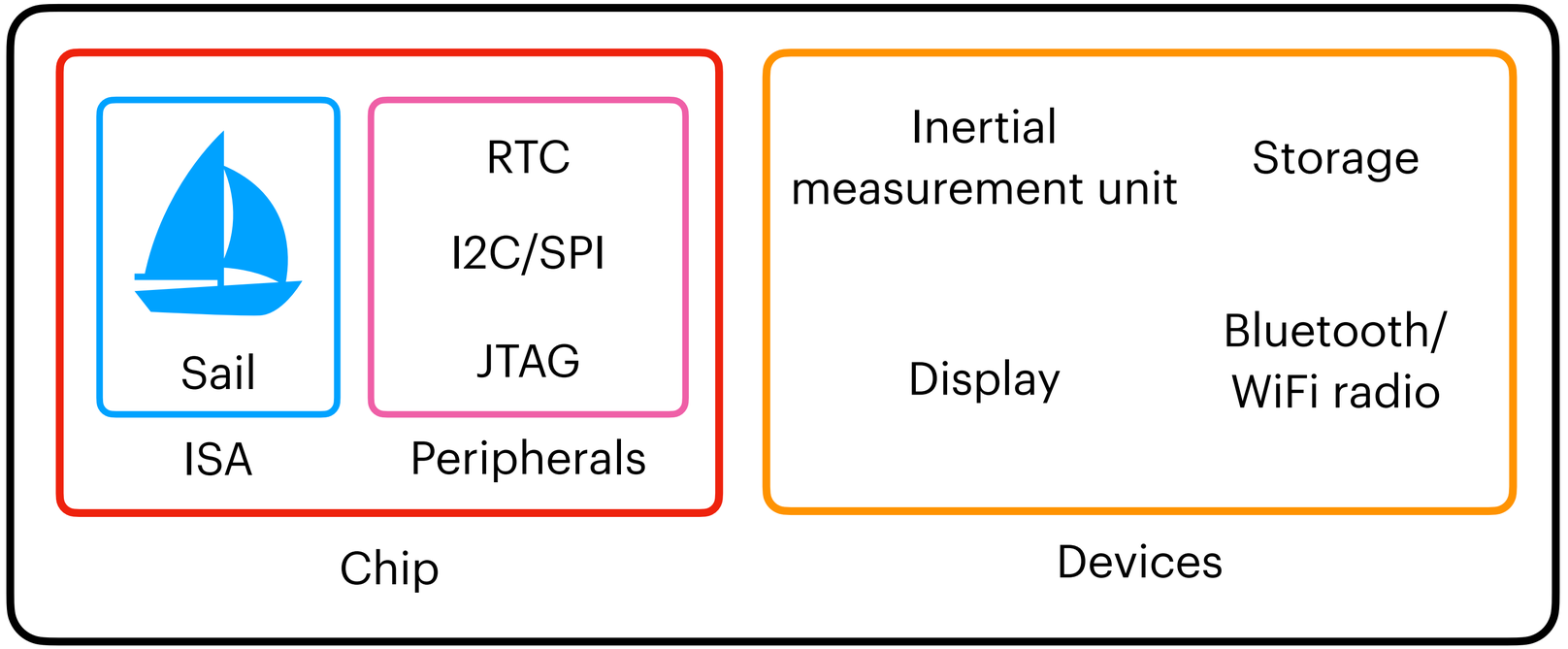
Instruction Execute



Instruction Execute



Future Research



Generating Fast Functional Simulators from Formal Specifications

2023 UK Systems Research Challenges Workshop

Ferdia McKeogh, Dr. Tom Spink, Prof. Al Dearle

fm208@st-andrews.ac.uk