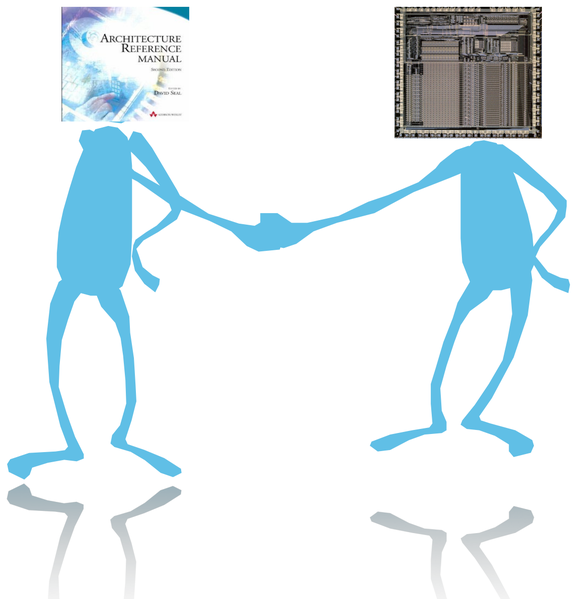
Hardware-Software Interfaces
Quality and Performance



Alastair Reid
Research
Arm Ltd

Aspects of hw/sw interface

Quality of specification

Performance

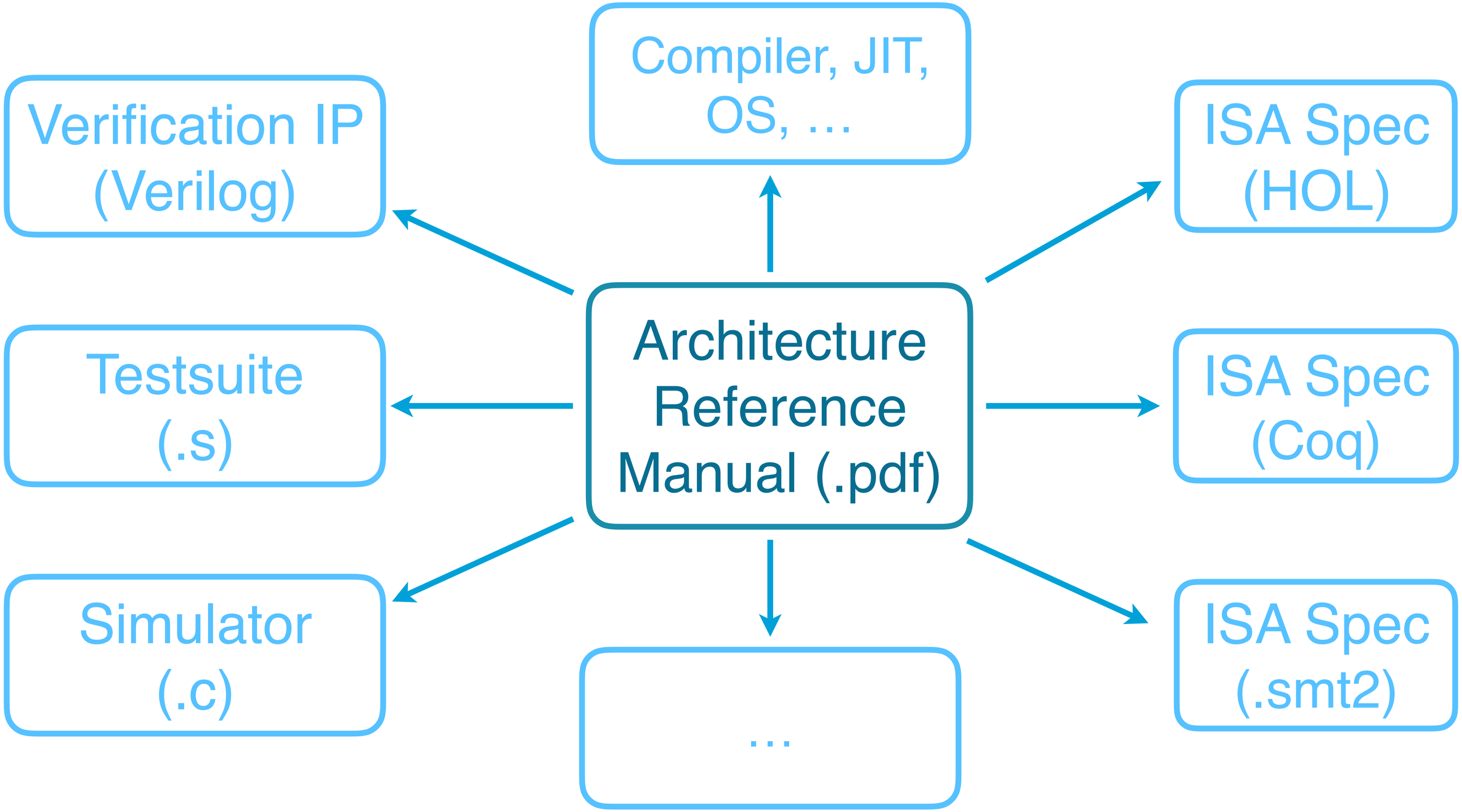
Security

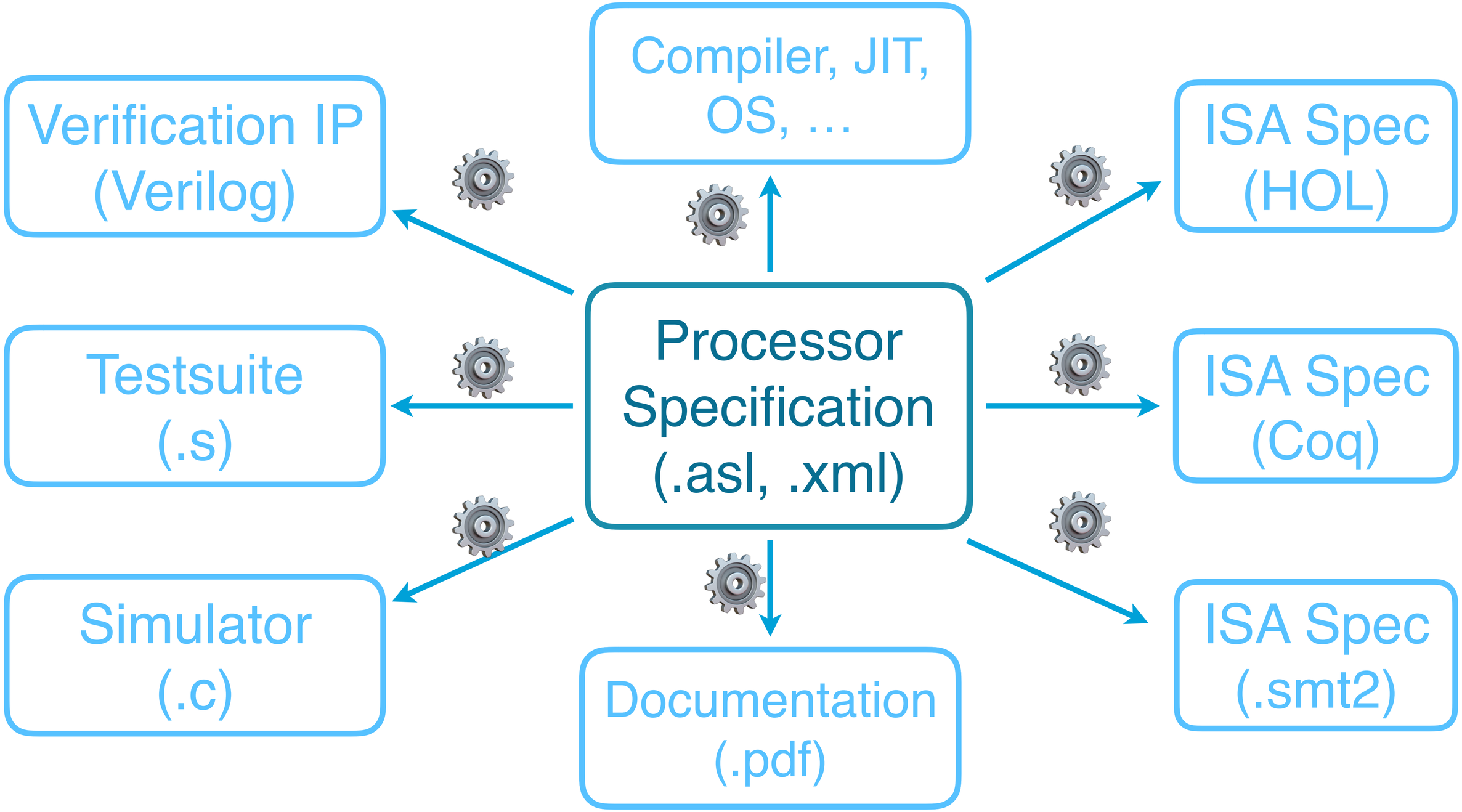
Scalability / Flexibility

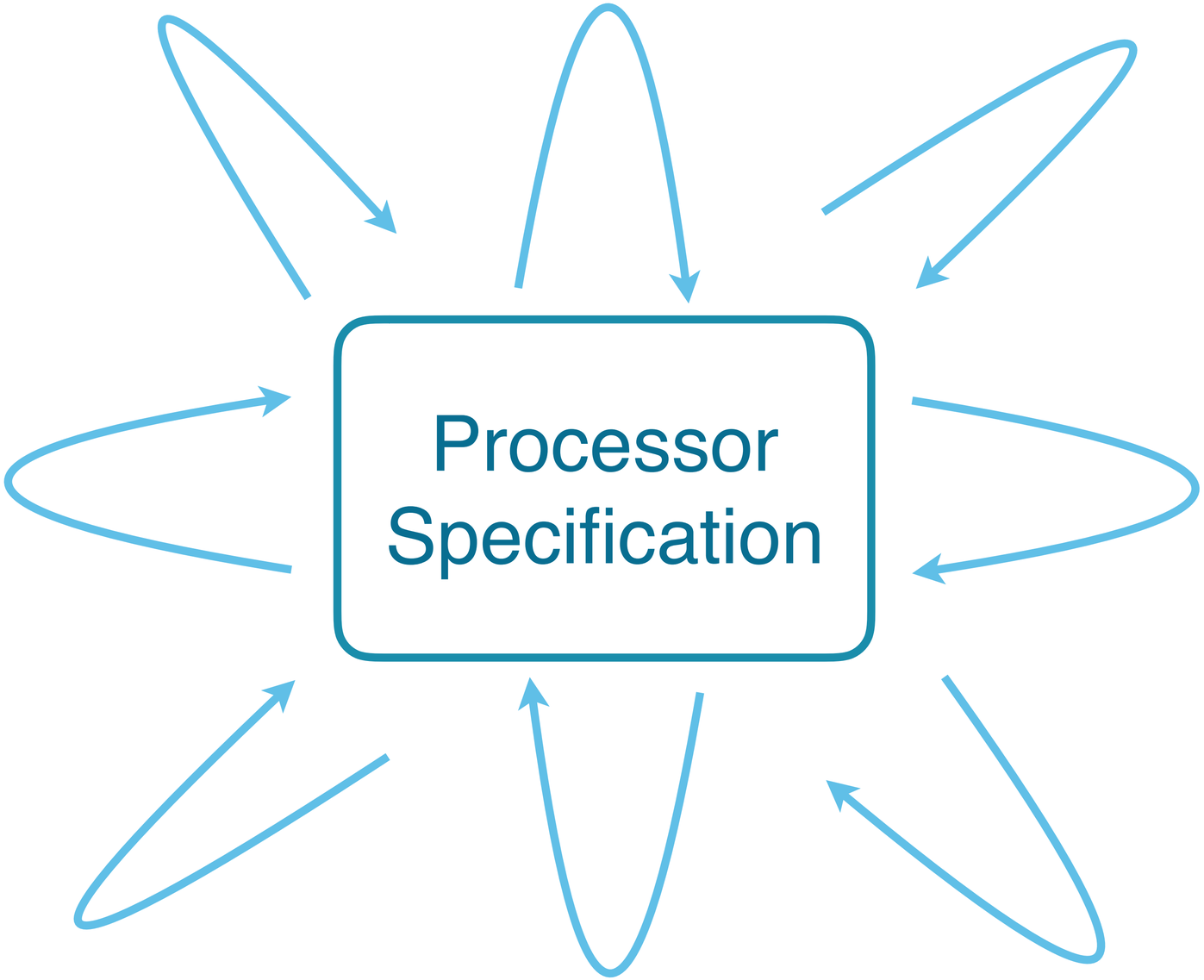
Parallelism

Energy efficiency

Area efficiency


Compiler, JIT, OS, …
Verification IP (Verilog)
ISA Spec (HOL)
Testsuite (.s)
Architecture Reference Manual (.pdf)
ISA Spec (Coq)
Simulator (.c)
…
ISA Spec (.smt2)


Compiler, JIT, OS, …
Verification IP (Verilog)
ISA Spec (HOL)
Testsuite (.s)
Processor Specification (.asl, .xml)
ISA Spec (Coq)
Simulator (.c)
ISA Spec (.smt2)
Documentation (.pdf)


Processor
Specification

Challenges

Increased requirements

What specification language?

Loss of redundancy - all eggs in one basket

Creating Specifications

Concurrent modification and execution of instructions

The ARMv8 architecture limits the set of instructions that can be executed by one thread of execution as they are being modified by another thread of execution without requiring explicit synchronization.

Concurrent modification and execution of instructions can lead to the resulting instruction performing any behavior that can be achieved by executing any sequence of instructions that can be executed from the same Exception level, except where each of the instruction before modification and the instruction after modification is one of a `B`, `BL`, `BRK`, `HVC`, `ISB`, `NOP`, `SMC`, or `SVC` instruction.

For the `B`, `BL`, `BRK`, `HVC`, `ISB`, `NOP`, `SMC`, and `SVC` instructions the architecture guarantees that, after modification of the instruction, behavior is consistent with execution of either:

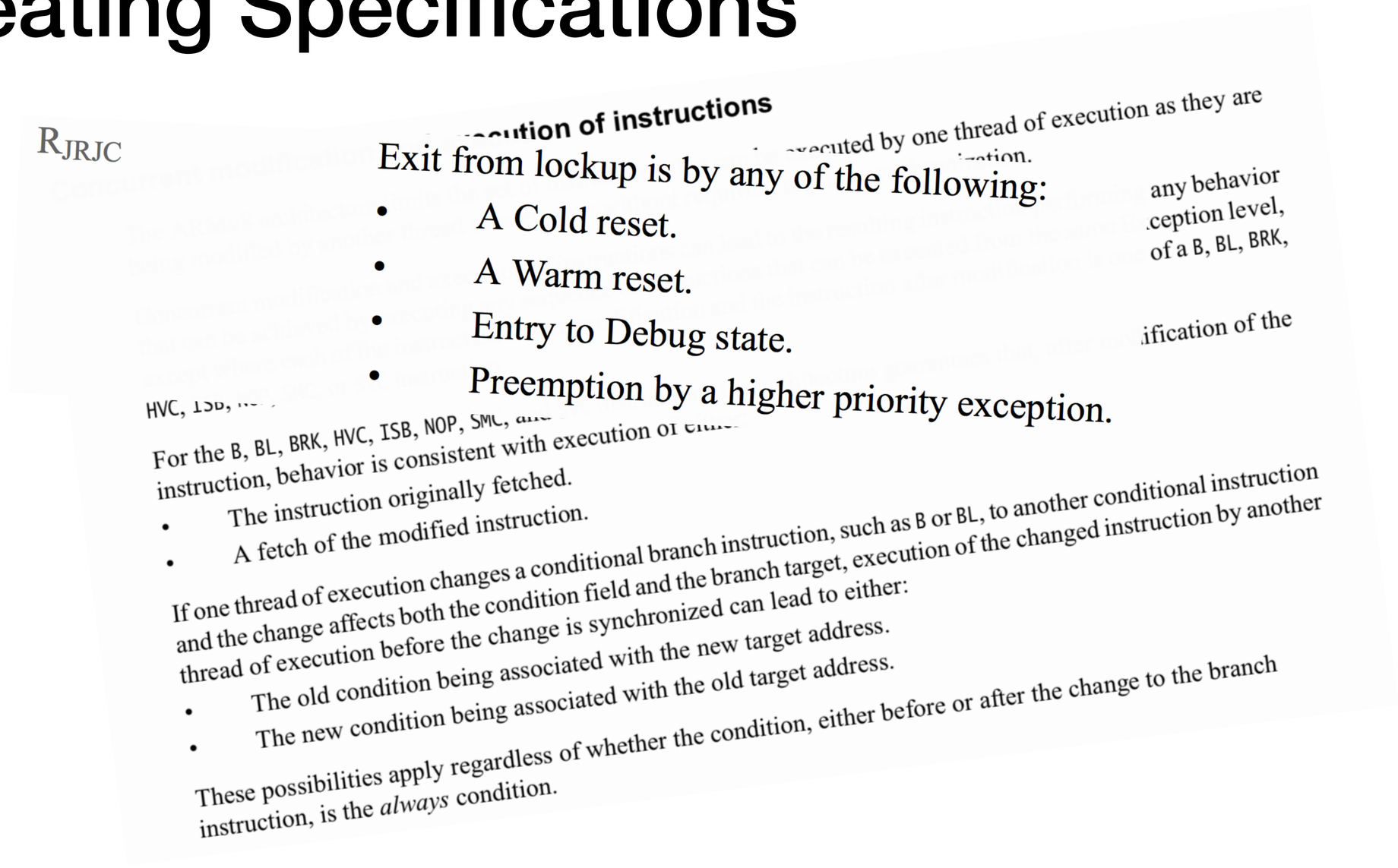
- The instruction originally fetched.
- A fetch of the modified instruction.

If one thread of execution changes a conditional branch instruction, such as `B` or `BL`, to another conditional instruction and the change affects both the condition field and the branch target, execution of the changed instruction by another thread of execution before the change is synchronized can lead to either:

- The old condition being associated with the new target address.
- The new condition being associated with the old target address.

These possibilities apply regardless of whether the condition, either before or after the change to the branch instruction, is the *always* condition.

Creating Specifications



RJRJC

...ecution of instructions

...executed by one thread of execution as they are

...ization.

Exit from lockup is by any of the following:

- A Cold reset.
- A Warm reset.
- Entry to Debug state.
- Preemption by a higher priority exception.

any behavior

...ception level,

of a B, BL, BRK,

HVC, ISB, ...

...ification of the

For the B, BL, BRK, HVC, ISB, NOP, SMC, ... instruction, behavior is consistent with execution of either:

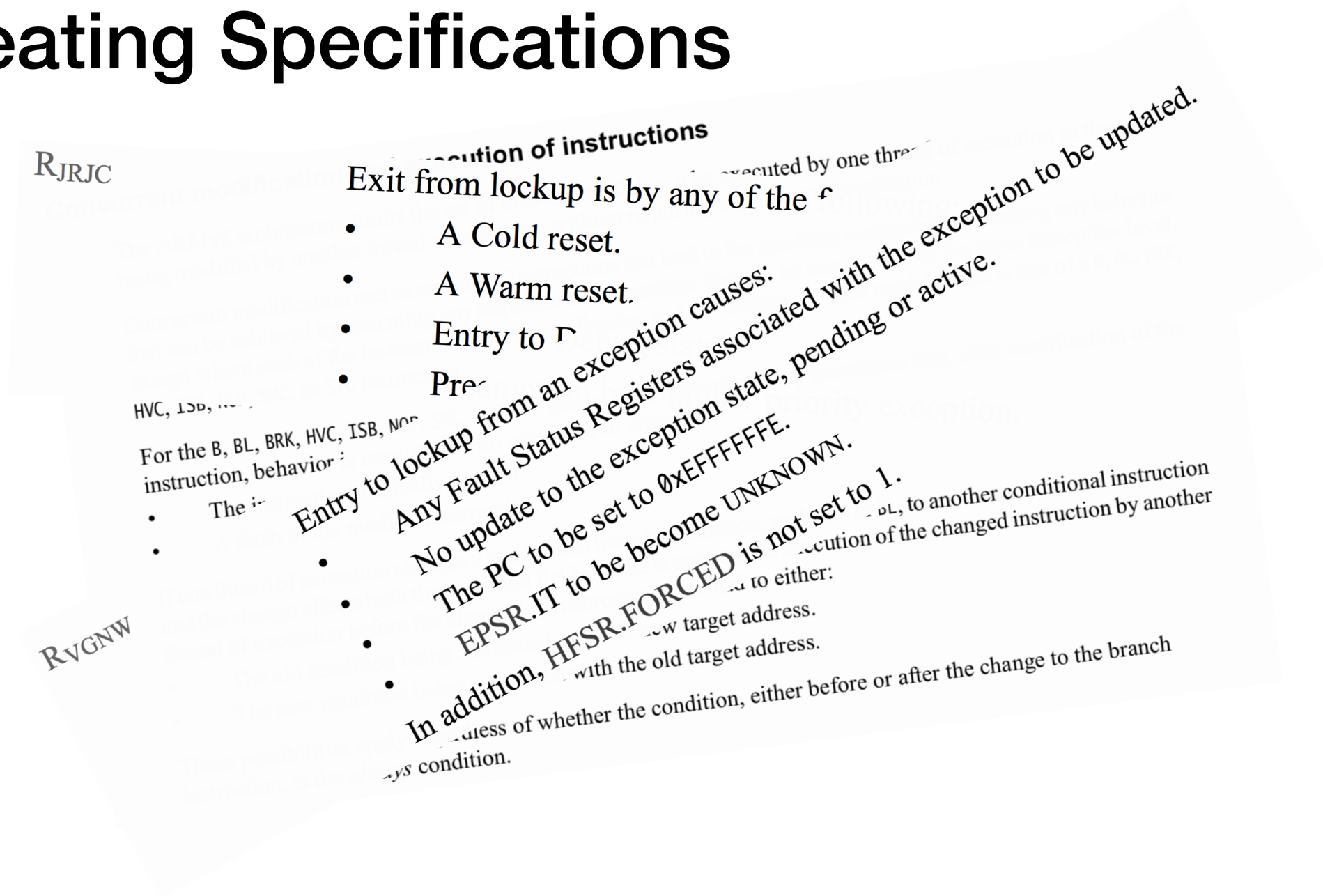
- The instruction originally fetched.
- A fetch of the modified instruction.

If one thread of execution changes a conditional branch instruction, such as B or BL, to another conditional instruction and the change affects both the condition field and the branch target, execution of the changed instruction by another thread of execution before the change is synchronized can lead to either:

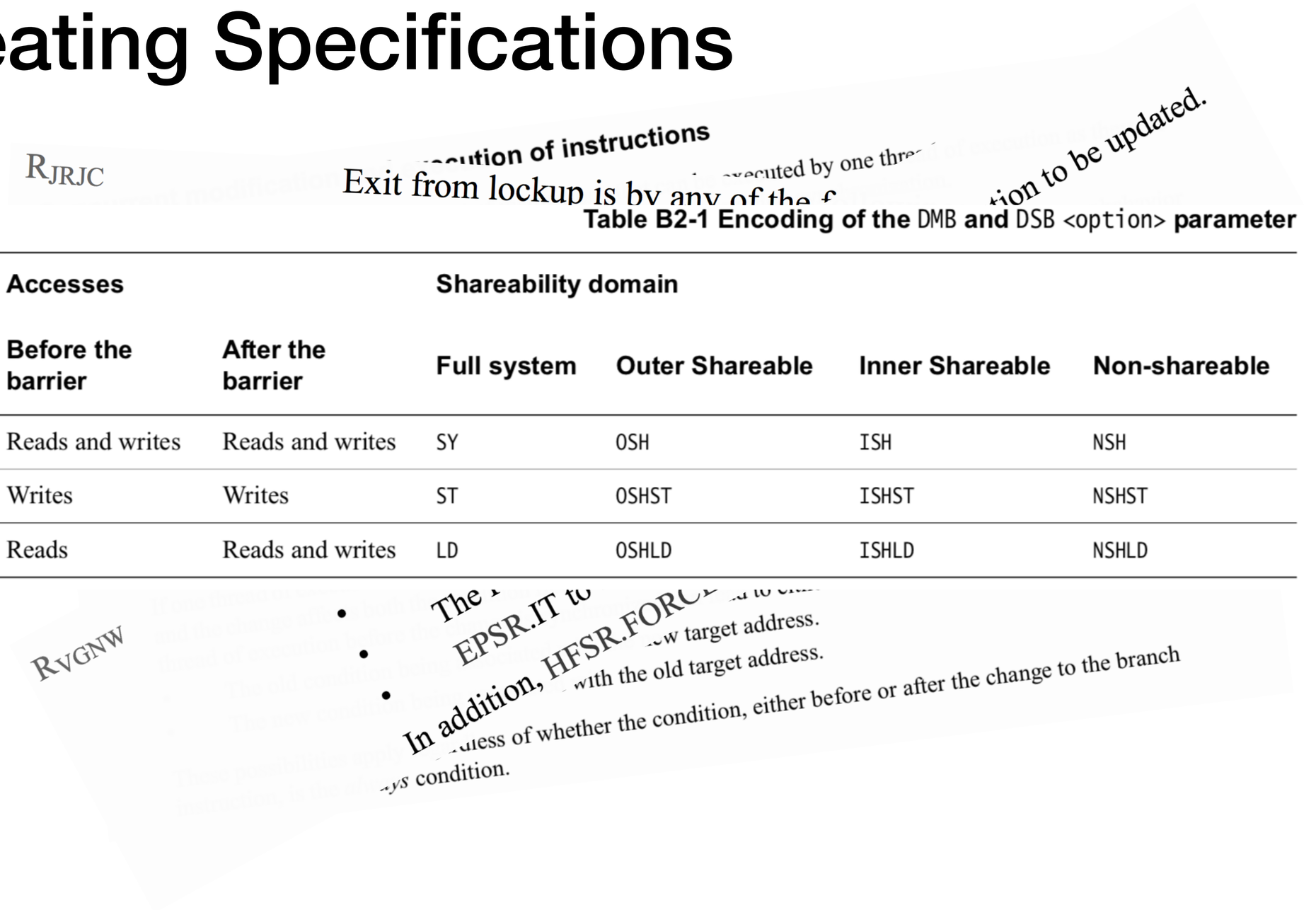
- The old condition being associated with the new target address.
- The new condition being associated with the old target address.

These possibilities apply regardless of whether the condition, either before or after the change to the branch instruction, is the *always* condition.

Creating Specifications



Creating Specifications



RJRJC

ecution of instructions

Exit from lockup is by any of the f

executed by one thre

tion to be updated.

Table B2-1 Encoding of the DMB and DSB <option> parameter

Accesses		Shareability domain			
Before the barrier	**After the barrier**	**Full system**	**Outer Shareable**	**Inner Shareable**	**Non-shareable**
Reads and writes	Reads and writes	SY	OSH	ISH	NSH
Writes	Writes	ST	OSHST	ISHST	NSHST
Reads	Reads and writes	LD	OSHLD	ISHLD	NSHLD

RVGNW

The

EPSR.IT to

In addition, HFSR.FORC

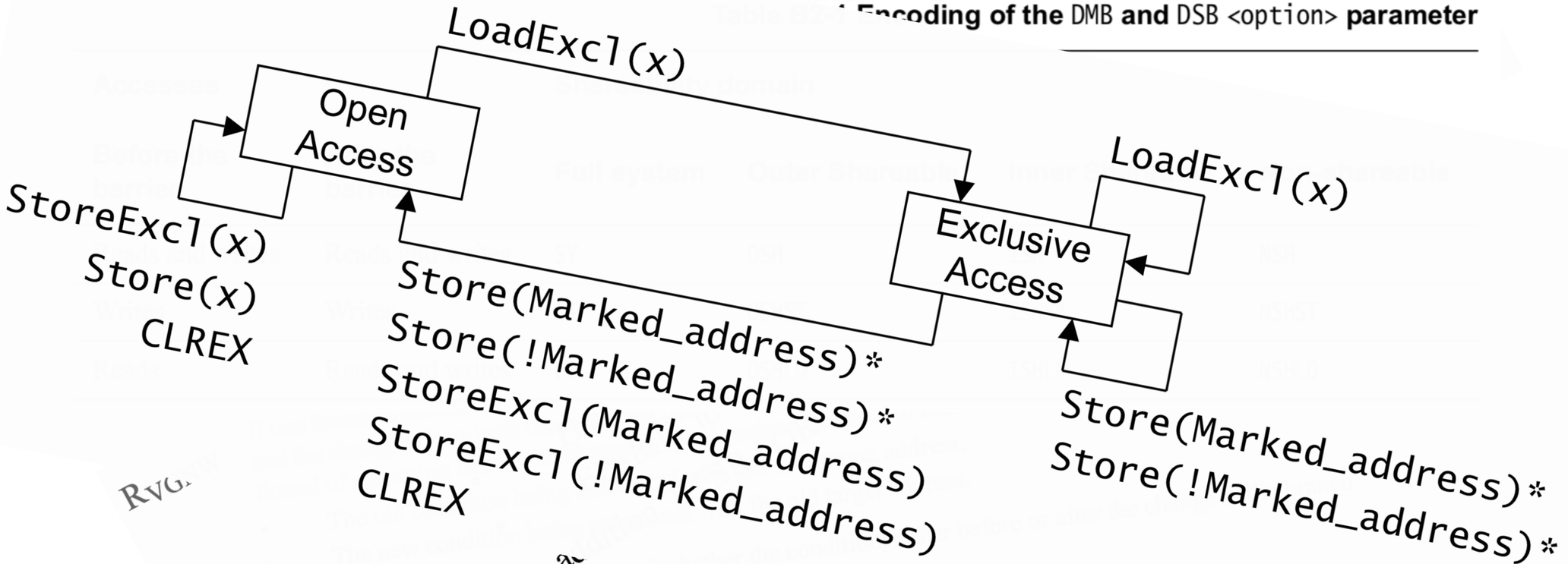
w target address.

with the old target address.

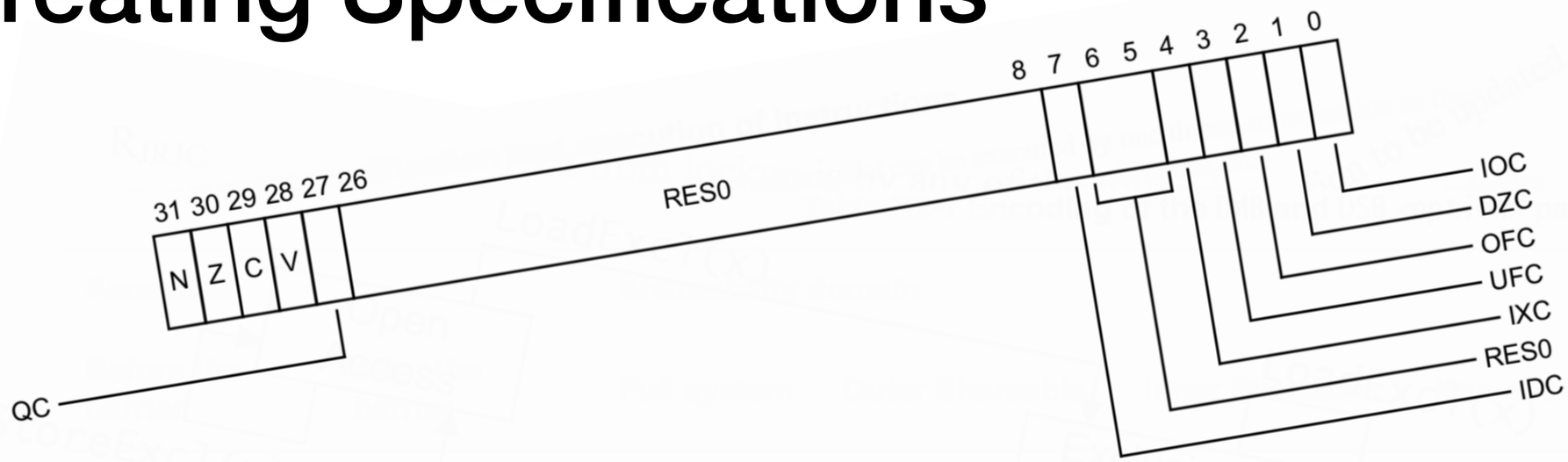
dless of whether the condition, either before or after the change to the branch

ys condition.

Creating Specifications



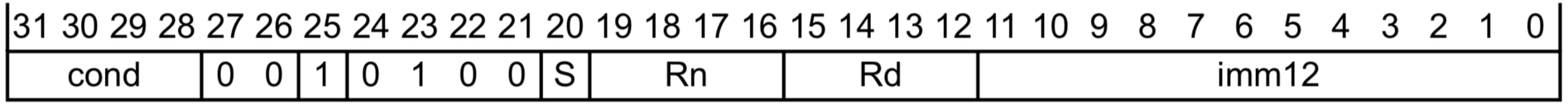
Creating Specifications



N, bit [31] Negative condition flag for AArch32 floating-point comparison operations. AArch64 floating-point comparisons set the PSTATE.N flag instead.

Z, bit [30] Zero condition flag for AArch32 floating-point comparison operations. AArch64 floating-point comparisons set the PSTATE.Z flag instead.

Pseudocode



For the case when cond is 0b1111, see *Unconditional instructions* on page A5-214.

```
if Rn == '1111' && S == '0' then SEE ADR;
if Rn == '1101' then SEE ADD (SP plus immediate);
if Rd == '1111' && S == '1' then SEE SUBS PC, LR and related instructions;
d = UInt(Rd);  n = UInt(Rn);  setflags = (S == '1');  imm32 = ARMExpandImm(imm12);
if ConditionPassed() then
    EncodingSpecificOperations();
    (result, carry, overflow) = AddWithCarry(R[n], imm32, '0');
    if d == 15 then
        ALUWritePC(result);  // setflags is always FALSE here
    else
        R[d] = result;
        if setflags then
            APSR.N = result<31>;
            APSR.Z = IsZeroBit(result);
            APSR.C = carry;
            APSR.V = overflow;
```

Arm Pseudocode

~40,000 lines

- 32-bit and 64-bit modes
- All 4 encodings: Thumb16, Thumb32, ARM32, ARM64
- All instructions (> 1300 encodings)
- All 4 privilege levels (User, Supervisor, Hypervisor, Secure Monitor)
- Both Security modes (Secure / NonSecure)
- MMU, Exceptions, Interrupts, Privilege checks, Debug, TrustZone, …

Status at the start

- Vague, incomplete, inaccurate language description
- No tools (parser, type checker)
- Incomplete (around 15% missing)
- Unexecuted, untested
- Senior architects believed that an executable spec was
  - Impossible
  - Not useful
  - Less readable
  - Less correct

Architectural Conformance Suite

Processor architectural compliance sign-off

Large

- v8-A 32,000 test programs, billions of instructions
- v8-M 3,500 test programs, > 250 million instructions

Thorough

- Tests dark corners of specification

Hard to run

- Requires additional testing infrastructure

Progress testing Arm specification

Does not parse, does not type check

Can’t get out of reset

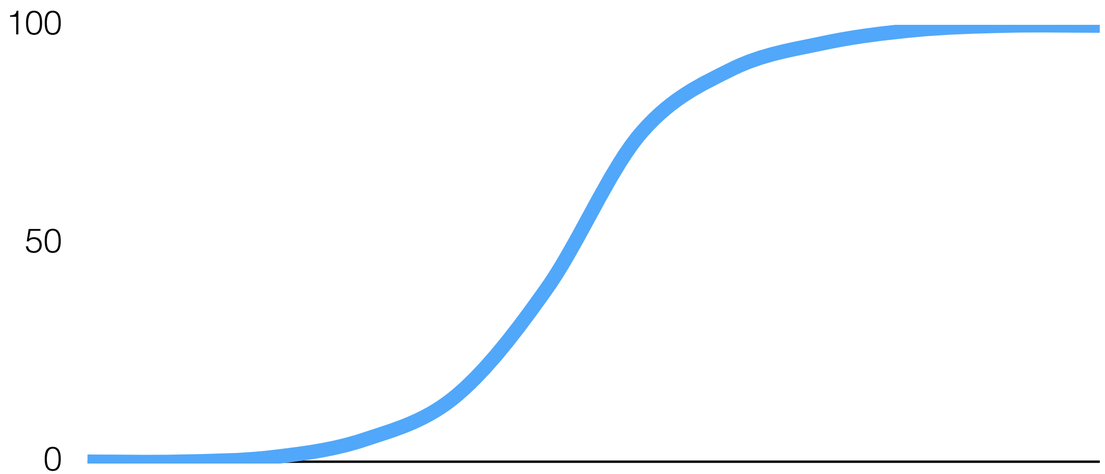
Can’t execute first instruction

Can’t execute first 100 instructions

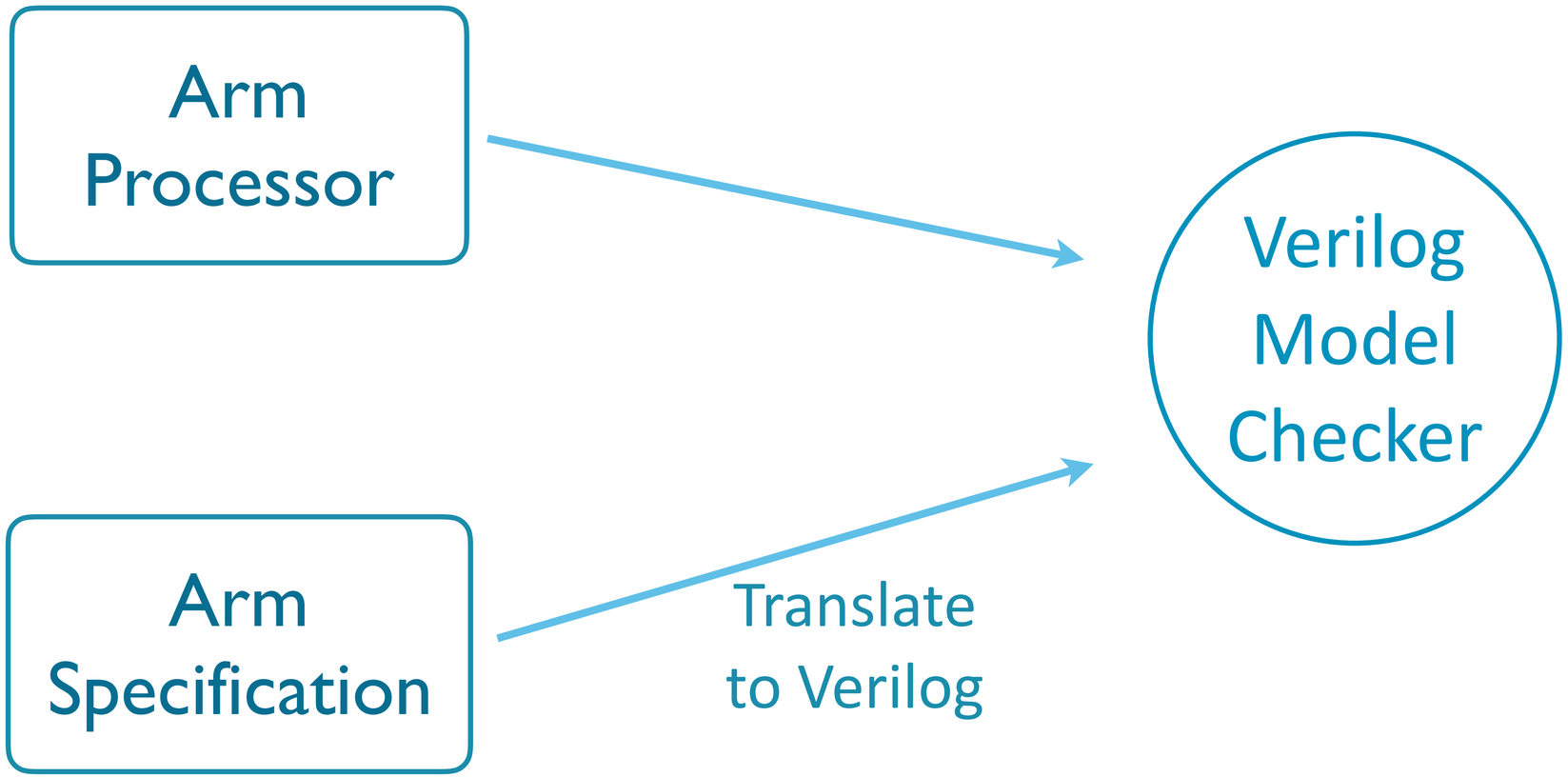
…

Passes 90% of tests

Passes 99% of tests



Formally validating Arm processors



Arm Specification Language

Verilog

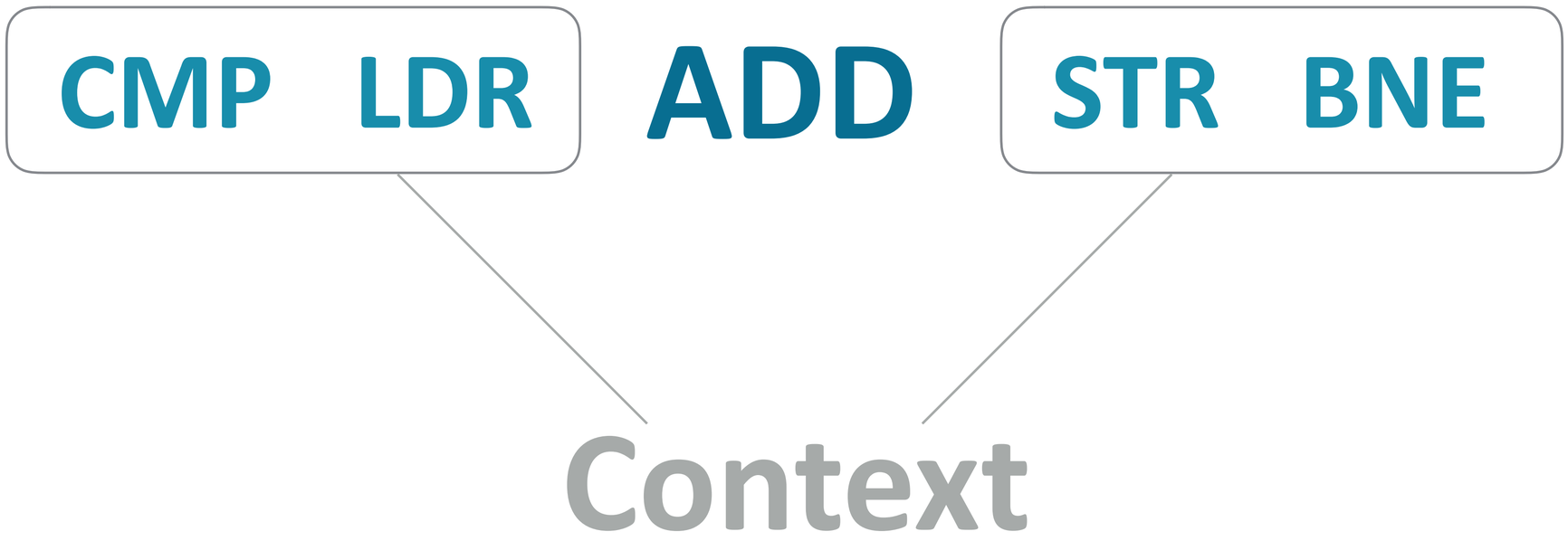
- Arithmetic operations
- Boolean operations
- Bit Vectors
- **Unbounded integers**
- Arrays
- **Dependent Types**
- Statements
  - Assignments
  - If-statements
  - **Loops**
  - **Exceptions**

- Arithmetic operations
- Boolean operations
- Bit Vectors
- ~~**Unbounded integers**~~
- Arrays
- ~~**Dependent Types**~~
- Statements
  - Assignments
  - If-statements
  - ~~**Loops**~~
  - ~~**Exceptions**~~

Checking an instruction

ADD

Checking an instruction



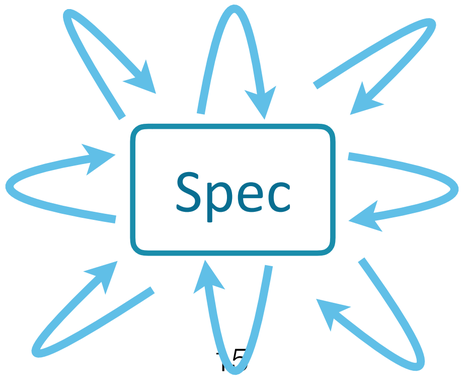
Lessons Learned from validating processors

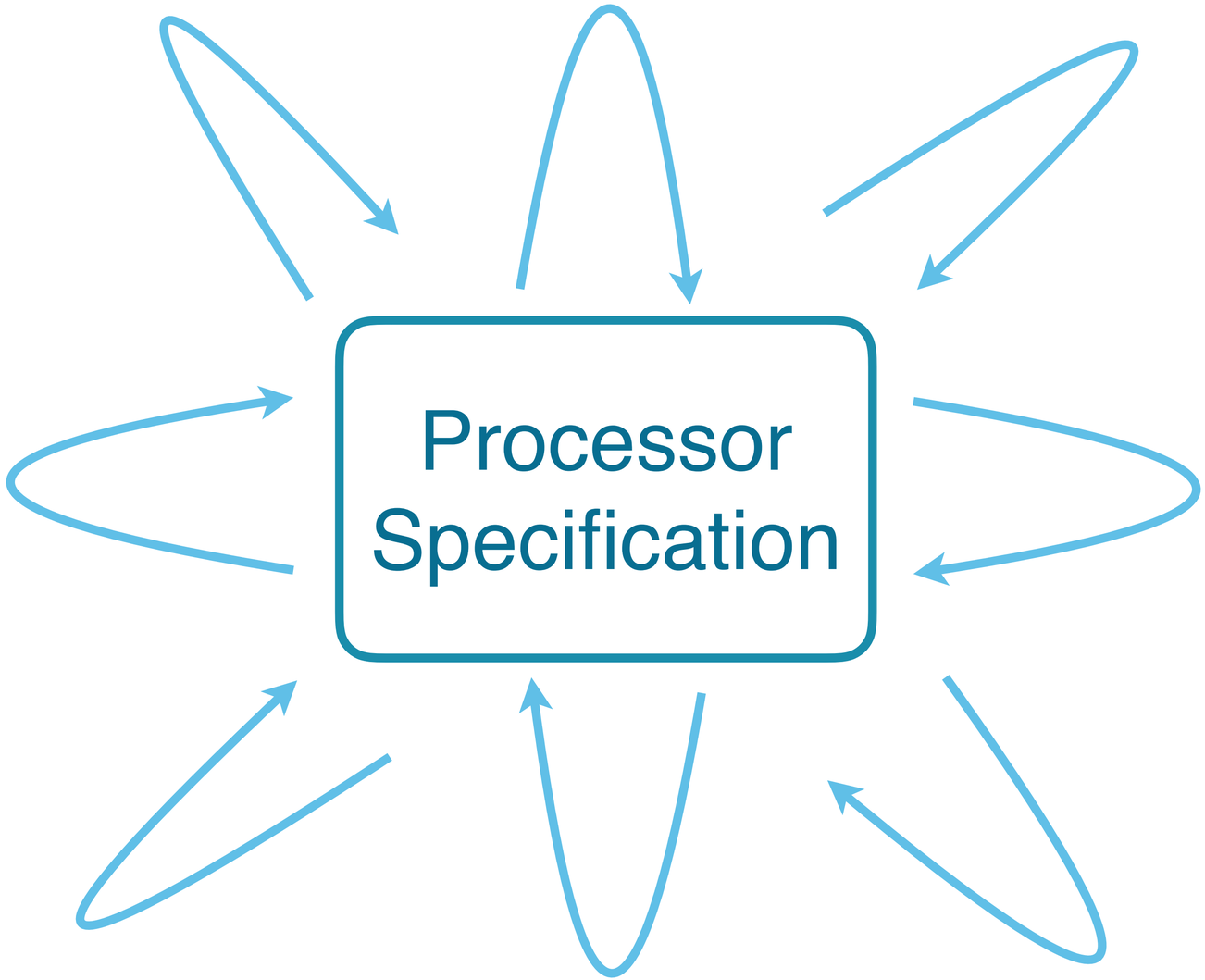
Very effective way to find bugs in implementations

Applied to commercial processors

- Cortex-A32, Cortex-A35, Cortex-A53, Cortex-A55, Cortex-A65
- Cortex-R52
- Cortex-M4, Cortex-M7, Cortex-M33

Formally validating implementations is effective at finding bugs in spec




Processor
Specification

Rule JRJC

Exit from lockup is by any of the following:

- A Cold reset.
- A Warm reset.
- Entry to Debug state.
- Preemption by a higher priority processor exception.

Rule R

State Change X by any of the following:

- Event A
- Event B
- State Change C
- Event D

Rule R

State Change X by any of the following:

- Event A
- Event B
- State Change C
- Event D

Rule R: $X \rightarrow A \vee B \vee C \vee D$

State Change X	Exit from lockup	Fell(LockedUp)
Event A	A Cold reset	Called(TakeColdReset)
Event B	A Warm reset	Called(TakeReset)
State Change C	Entry to Debug state	Rose(Halted)
Event D	Preemption by a higher priority processor exception	Called(ExceptionEntry)

$$\text{Fell(LockedUp)} \rightarrow \text{Called(TakeColdReset)} \vee \text{Called(TakeReset)} \vee \text{Rose(Halted)} \vee \text{Called(ExceptionEntry)}$$

Rule JRJC

Exit from lockup is by any of the following:

- A Cold reset.
- A Warm reset.
- Entry to Debug state.
- Preemption by a higher priority processor exception.

Arm Specification Language

SMT

- Arithmetic operations
- Boolean operations
- Bit Vectors
- Arrays
- Dependent Types
- Functions
- Statements
  - Assignments
  - If-statements
  - Loops
  - Exceptions

- Arithmetic operations
- Boolean operations
- Bit Vectors
- ~~Arrays~~
- ~~**Dependent Types**~~
- ~~Functions~~
- ~~**Statements**~~
  - ~~**Assignments**~~
  - ~~**If-statements**~~
  - ~~**Loops**~~
  - ~~**Exceptions**~~

Results

Most properties proved in under 100 seconds (each)

Found 12 bugs in specification:

- debug, exceptions, system registers, security

Found bugs in English prose:

- ambiguous, imprecise, incorrect, …

Public release of Arm v8-A specification

Enable formal verification of software and tools

Machine readable

Up to date (v8.5-A architecture released Sept'18)

Automatic translation to Sail and Isabelle with Cambridge University

- Rerun Architecture Conformance Suite
- "ISA Semantics for ARM v8-A, , RISC-V, and CHERI-MIPS," POPL 2019

https://developer.arm.com/products/architecture/a-profile/exploration-tools
https://github.com/alastairreid/mra_tools
https://github.com/rems-project/sail-arm

Summary

Mechanized major commercial architecture specification

- High quality, broad scope

Formal validation of processors

- Applied to multiple commercial processors
- Adapted for use with other architectures

Formal validation of processor specification

Public release of specification

- Translation of Arm's specification to Sail enabling Isabelle proofs w/ Cambridge

"Trustworthy Specifications of the ARM v8-A and v8-M architecture" FMCAD 2016
"End to End Verification of ARM processors with ISA Formal" CAV 2016
"Who guards the guards? Formal Validation of ARM v8-M Specifications" OOPSLA 2017
["ISA Semantics for ARM v8-A, RISC-V, and CHERI-MIPS" POPL 2019]

Performance

Software

Instruction Stream

Hardware

Performance

Sequential Software

Sequential Instruction Stream

Parallel Hardware

Performance

Sequential Software

Sequential Instruction Stream

Out-of-Order Execution

Parallel Hardware

Performance

Sequential Software

Parallel Instruction Streams

Parallel Hardware

Performance

Sequential Software

Parallelizing Compiler

Parallel Instruction Streams

Parallel Hardware

Performance

Annotations

+

Sequential Software

Parallelizing Compiler

Parallel Instruction Streams

Parallel Hardware

Performance

Annotations

+

Sequential Software

Parallelizing Compiler

Parallel Hardware

Challenges

Annotations depend on features of hardware and domain

Programmer burden

Compiler transformations

Ardbeg (2006-2008)

Goal: Build a commercial software defined radio system for LTE protocol

Subsystem:

- 2-4x 450MHz VLIW processors
  - 512-bit predicated SIMD
  - 14.4 Gops @ 250mW (each)
- Custom accelerators (Viterbi, Turbo, …)

Asymmetric MP

Heterogeneous

Specialized cores

Local memories per processor

Explicit data copying (DMA)

Symmetric MP

Homogeneous

General Purpose cores

Cache hierarchy

Cache coherence

Hardware + Requirements

Heterogeneous

Specialized cores

Local memories

Explicit data copying (DMA)

Language Features

Pipeline Parallelism

Explicit function placement (RPC)

Explicit data placement

Static Distributed Shared Memory

Programmer in control

Portability

Rapid design space exploration

Don’t hide expensive operations

Annotations direct restructuring

Annotation checking and inference

```
int x[100];
int y[100];
int z[100];

while (1) {
        get(x);
        foo(y,x);


        bar(z,y);
        baz(z);
        put(z);
}
```

```
int x[100];
int y[100];
int z[100];

while (1) {
        get(x);
        foo(y,x) @ P0;
        SYNC(x) @ DMA;

        bar(z,y) @ P1;
        baz(z) @ P1;
        put(z);
}
```

Synchronize data

Remote Procedure Call

Use pipeline Parallelism

Transfer data between threads

```
int x[100];
int y[100];
int z[100];
PIPELINE {
  while (1) {
      get(x);
      foo(y,x) @ P0;
      SYNC(x) @ DMA;
      FIFO(y);
      bar(z,y) @ P1;
      baz(z) @ P1;
      put(z);
  }
}
```

```
int x[100] @ {M0};
int y[100] @ {M0,M1};
int z[100] @ {M1};
PIPELINE {
  while (1) {
      get(x@M0);
      foo(y@M0, x@M0) @ P0;
      SYNC(y,M1,M0) @ DMA;
      FIFO(y@M1);
      bar(z@M1, y@M1) @ P1;
      baz(z@M1) @ P1;
      put(z@M1);
  }
}
```

```
int x[100] @ {M0};
int y0[100] @ {M0};
int y1a[100] @ {M1};
while (1) {
   get(x);
   foo(y0, x) @ P0;
   memcpy(y1a,y0,…) @ DMA;
   fifo_put(&f, y1a);
}
```

```
int y1b[100] @ {M1};
int z[100] @ {M1};
while (1) {
   fifo_get(&f, y1b);
   bar(z, y1b) @ P1;
   baz(z) @ P1;
   put(z);
}
```

Summary

Part of “Ardbeg” Software Defined Radio project

- Energy efficient LTE radio modem
- Competitive with fixed function hardware

Language extensions balance performance and portability

- Programmer uses annotations to control software-hardware mapping
- Compiler restructures program to implement annotations

Defining hw/sw interface at a higher level to enable greater performance

“SoC-C: Efficient Programming Abstractions for Heterogeneous Multicore Systems on Chip” CASES 2016
“Reducing inter-task latency in a multiprocessor system” US 8,359,588
[“From SODA to scotch: The evolution of a wireless baseband processor” MICRO 2018]

Hardware-Software Interfaces

Two aspects of interface explored

- High Quality Specifications
- Performance

Future work

- Security
  - Can we verify that ISA is “secure”?
- Parallelism
  - Can we apply ideas to other parallelism frameworks?

Fin