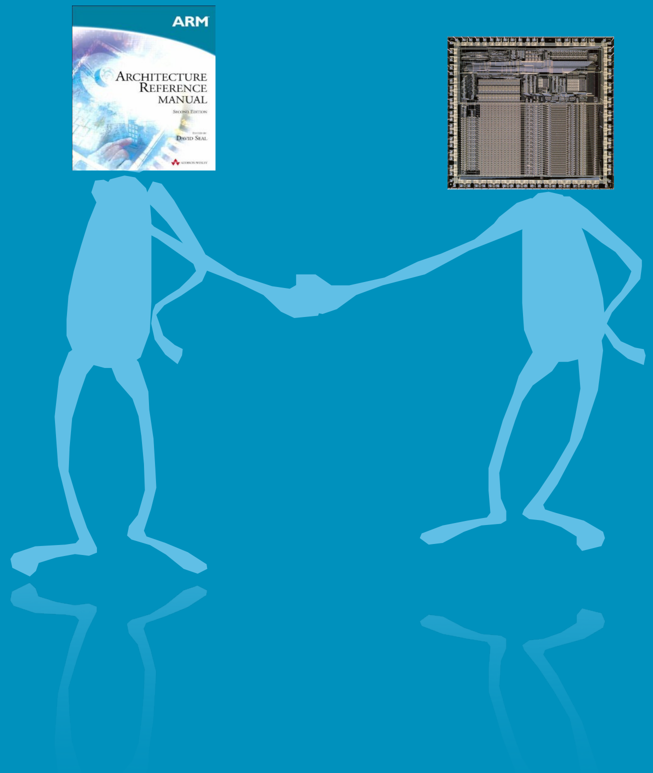
Specifications: The Next Verification Bottleneck

Alastair Reid

Arm Research

@alastair_d_reid



Overview

1. What specifications do we need?
2. ARM’s formal processor specifications
3. Three steps I took to create good specifications

ARM

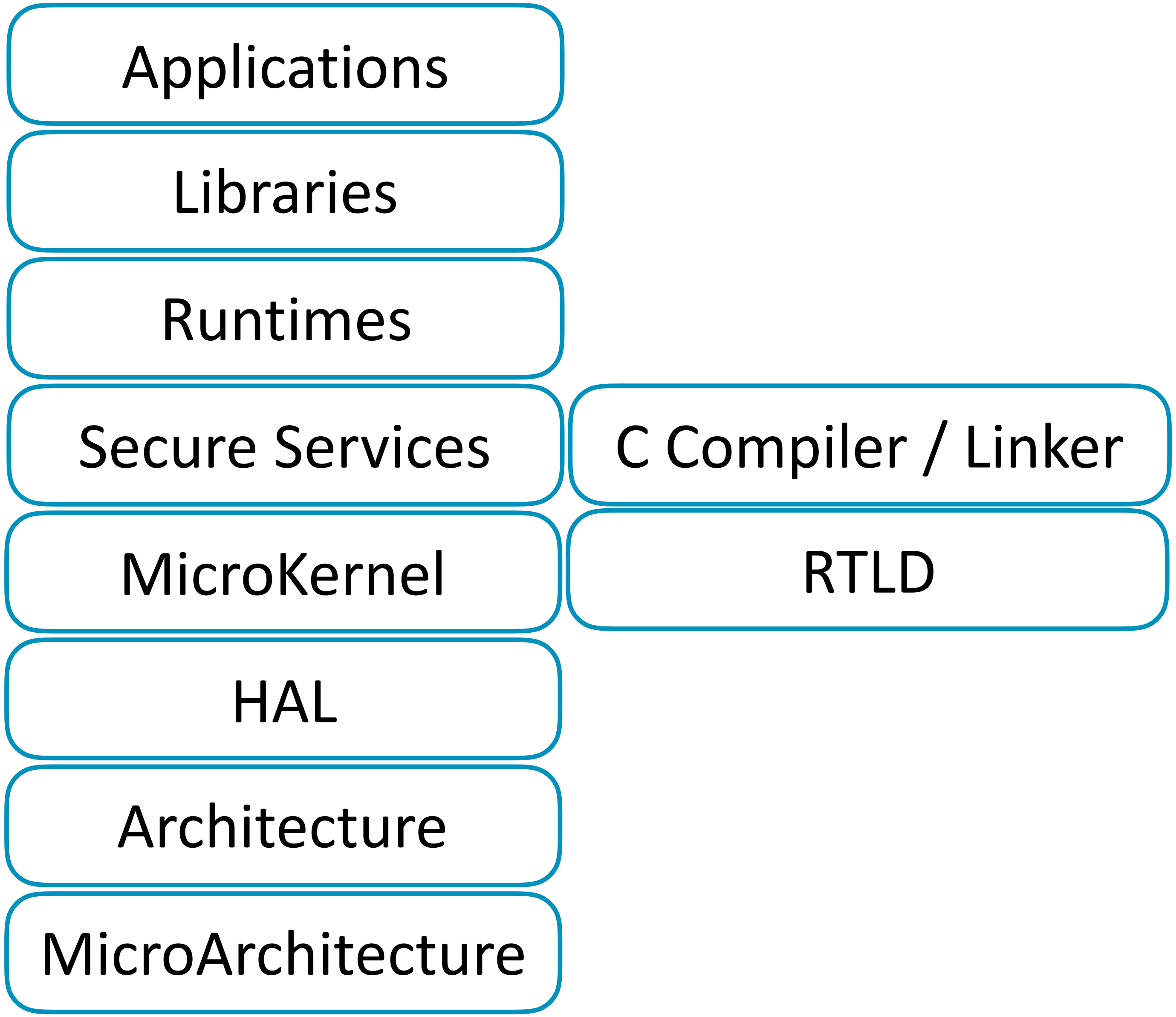
Designs processors, designs architecture, licenses architecture

16B processors / year

(also GPUs, IoT, …)

Security Research Group

- Develop and analyse security extensions
- Create framework for verifying products
- We are hiring: full time, research internships



Applications

Libraries

Runtimes

Secure Services

C Compiler / Linker

MicroKernel

RTLD

HAL

Architecture

MicroArchitecture

Specifications we need

Linux sys calls

C stdlib

Processor page tables

Interrupt handler

Device driver API

Filesystem format

ISO C

Gcc/LLVM extensions

Inline assembly

ELF / linkerscript

Weak memory model

TCP/IP, UDP, …

TSL

NTP, DNS, NFS, …

WiFi, Bluetooth, Zigbee, …

USB, SD card, …

X11/Gtk+/…

Javascript, CSS, SVG, …

PHP, …

Trusted Computing Base (!= Trustworthy Computing Base)

a small amount of software and hardware that security depends on and that we distinguish from a much larger amount that can misbehave without affecting security

— Lampson

the totality of protection mechanisms within it, including hardware, firmware, and software, the combination of which is responsible for enforcing a computer security policy

— Orange Book (US DoD)

Specifications for real world software/hardware

Unavoidable

Multiple implementations

Multiple versions of each implementation

Spec must include all quirks of recent versions of major implementations to be useful

Existing specification = English + Tables + Pseudocode

Existing community may not value formal spec at first

“Trustworthy Specifications of the ARM v8-A and v8-M architecture,” FMCAD 2016

Creating trustworthy specifications

The state of most processor specifications

Large (1000s of pages)

Broad (10+ years of implementations, multiple manufacturers)

Complex (exceptions, weak memory, …)

Informal (mostly English prose)

We are all just learning how to (retrospectively) formalize specifications

Arm Processor Specifications

A-class (phones, tablets, servers, …)

6,000 pages
40,000 line formal specification

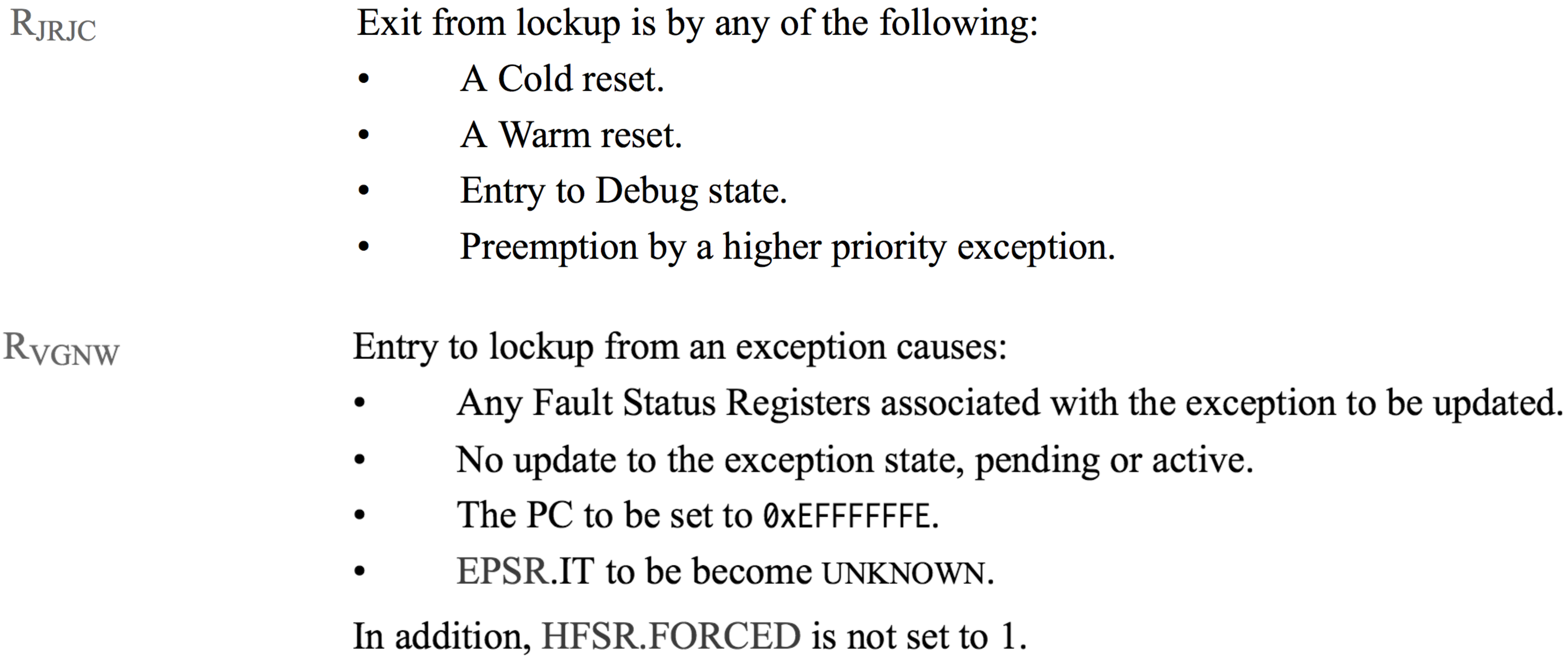
Instructions (32/64-bit)
Exceptions / Interrupts
Memory protection
Page tables
Multiple privilege levels
System control registers
Debug / trace

M-class (microcontrollers, IoT)

1,200 pages
15,000 line formal specification

Instructions (32-bit)
Exceptions / Interrupts
Memory protection
~~Page tables~~
Multiple privilege levels
System control registers
Debug / trace

English prose



R_{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 exception.

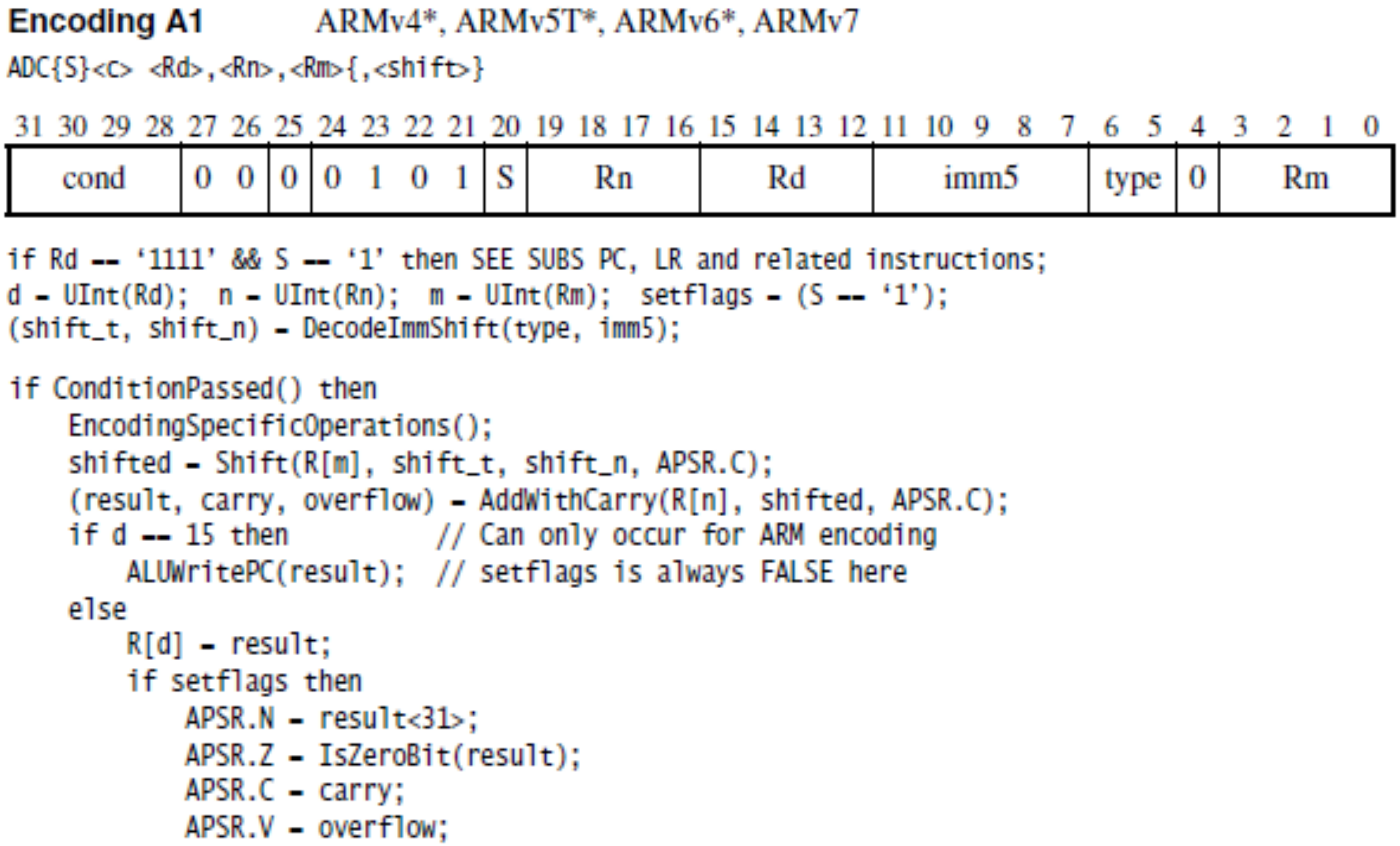
R_{VGNW}

Entry to lockup from an exception causes:

- Any Fault Status Registers associated with the exception to be updated.
- No update to the exception state, pending or active.
- The PC to be set to `0xEFFFFFFE`.
- EPSR.IT to be become UNKNOWN.

In addition, HFSR.FORCED is not set to 1.

Pseudocode



Encoding A1 ARMv4*, ARMv5T*, ARMv6*, ARMv7

`ADC{S}<c> <Rd>,<Rn>,<Rm>{,<shift>}`

31 30 29 28	27 26	25	24 23 22 21	20	19 18 17 16	15 14 13 12	11 10 9 8 7	6 5	4	3 2 1 0
cond	0 0	0	0 1 0 1	S	Rn	Rd	imm5	type	0	Rm

```
if Rd -- '1111' && S -- '1' then SEE SUBS PC, LR and related instructions;
d - UInt(Rd);  n - UInt(Rn);  m - UInt(Rm);  setflags - (S -- '1');
(shift_t, shift_n) - DecodeImmShift(type, imm5);

if ConditionPassed() then
    EncodingSpecificOperations();
    shifted - Shift(R[m], shift_t, shift_n, APSR.C);
    (result, carry, overflow) - AddWithCarry(R[n], shifted, APSR.C);
    if d -- 15 then          // Can only occur for ARM encoding
        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;
```

System Architecture Specification

```
AArch64.DataAbort(bits(64) vaddress, FaultRecord fault)

    route_to_el3 = HaveEL(EL3) && SCR_EL3.EA == '1' && IsExternalAbort(fault);
    route_to_el2 = (HaveEL(EL2) && !IsSecure() && PSTATE.EL IN {EL0,EL1} &&
                    (HCR_EL2.TGE == '1' || IsSecondStage(fault)));

    bits(64) preferred_exception_return = ThisInstrAddr();
    vect_offset = 0x0;

    exception = AArch64.AbortSyndrome(Exception_DataAbort, fault, vaddress);

    if PSTATE.EL == EL3 || route_to_el3 then
        AArch64.TakeException(EL3, exception, preferred_exception_return, vect_offset);
    elsif PSTATE.EL == EL2 || route_to_el2 then
        AArch64.TakeException(EL2, exception, preferred_exception_return, vect_offset);
    else
        AArch64.TakeException(EL1, exception, preferred_exception_return, vect_offset);
```

Arm Architecture Specification Language (ASL)

- Indentation-based syntax
- Imperative
- First-order
- Strongly typed (type inference, polymorphism, dependent types)
  - Bit-vectors
  - Unbounded integers
  - Infinite precision reals
  - Arrays, Records, Enumerations
- Exceptions

ARM Spec (lines of code)

	v8-A	v8-M
Instructions **Int/FP/SIMD**	26,000	6,000
Exceptions	4,000	3,000
Memory	3,000	1,000
Debug	3,000	1,000
Misc	5,500	2,000
(Test support)	1,500	2,000
Total	**43,000**	**15,000**

System Register Spec

	v8-A	v8-M
Registers	586	186
Fields	3951	622
Constant	985	177
Reserved	940	208
Impl. Defined	70	10
Passive	1888	165
Active	68	62
Operations	112	10

Trustworthiness

Trustworthiness

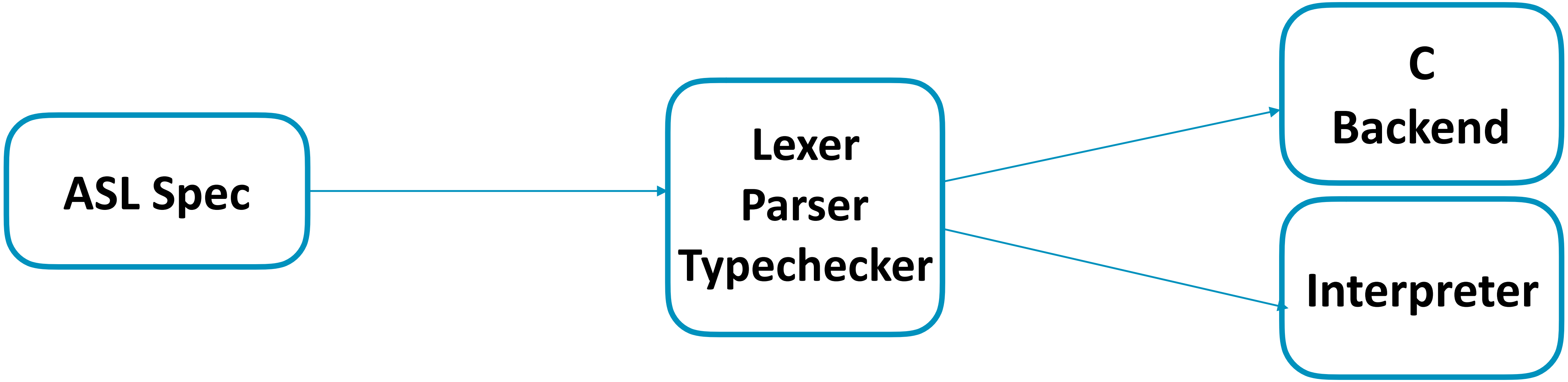
ARM's specification is correct *by definition*

Trustworthiness

~~ARM’s specification is correct *by definition*~~

Trustworthiness

Does the specification match the behaviour of all ARM processors?



ASL Spec

Lexer
Parser
Typechecker

C
Backend

Interpreter

Architectural Conformance Suite

Processor architectural compliance sign-off

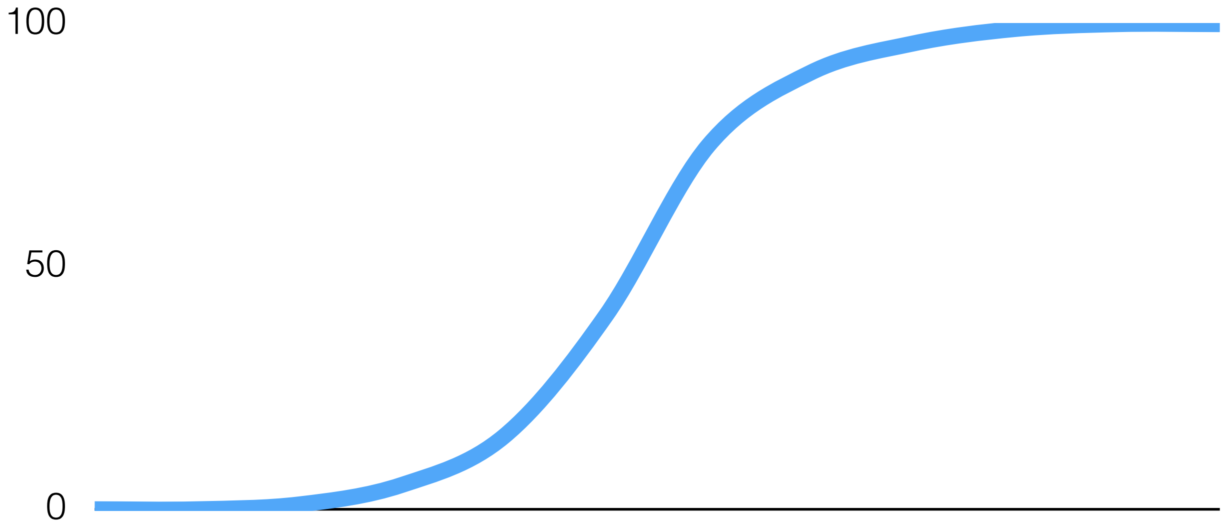
Large

- v8-A 11,000 test programs, > 2 billion instructions
- v8-M 3,500 test programs, > 250 million instructions

Thorough

- Tests dark corners of specification

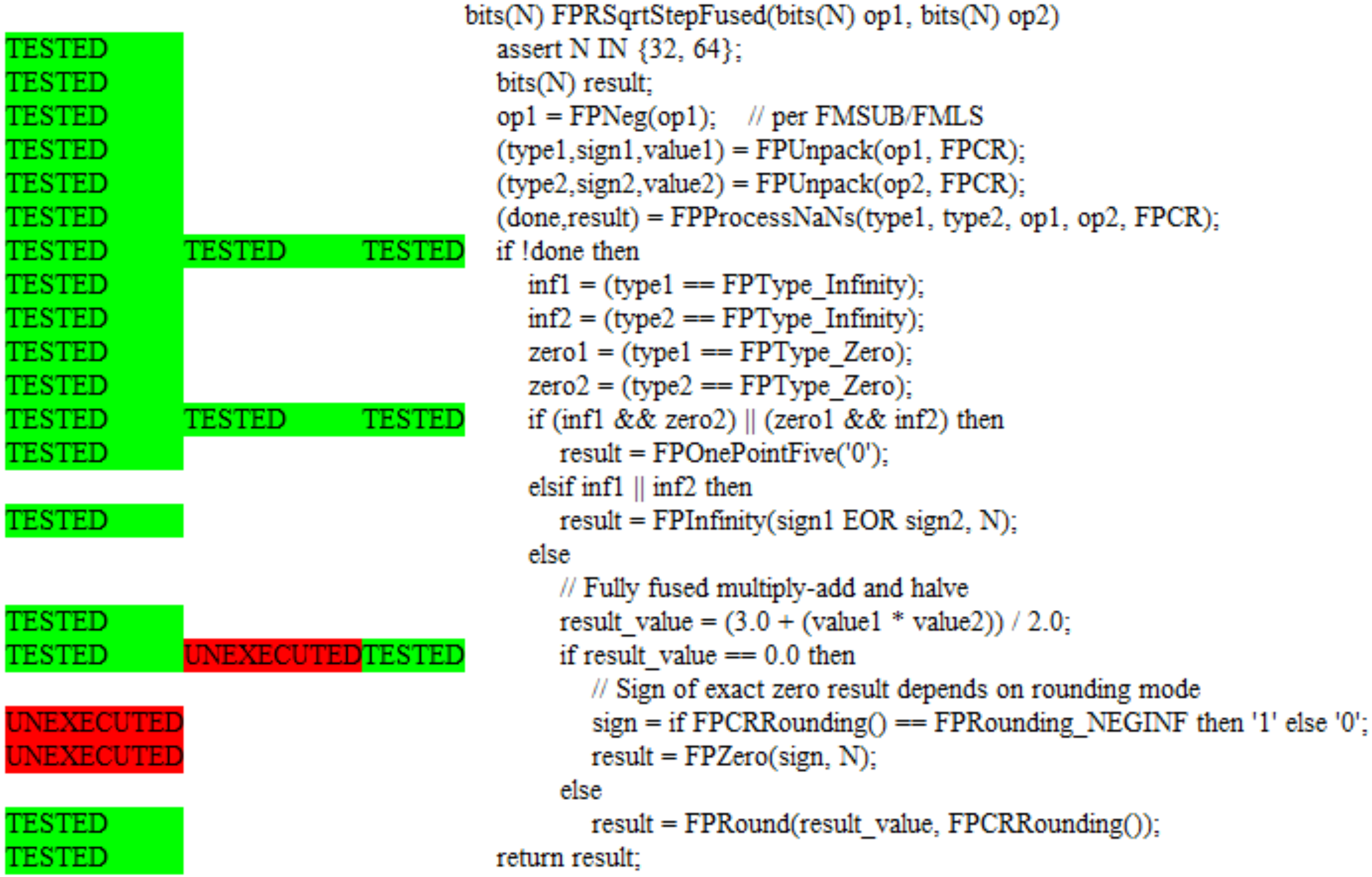
Progress in testing Arm specification



- Does not parse, does not typecheck
- 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
- …

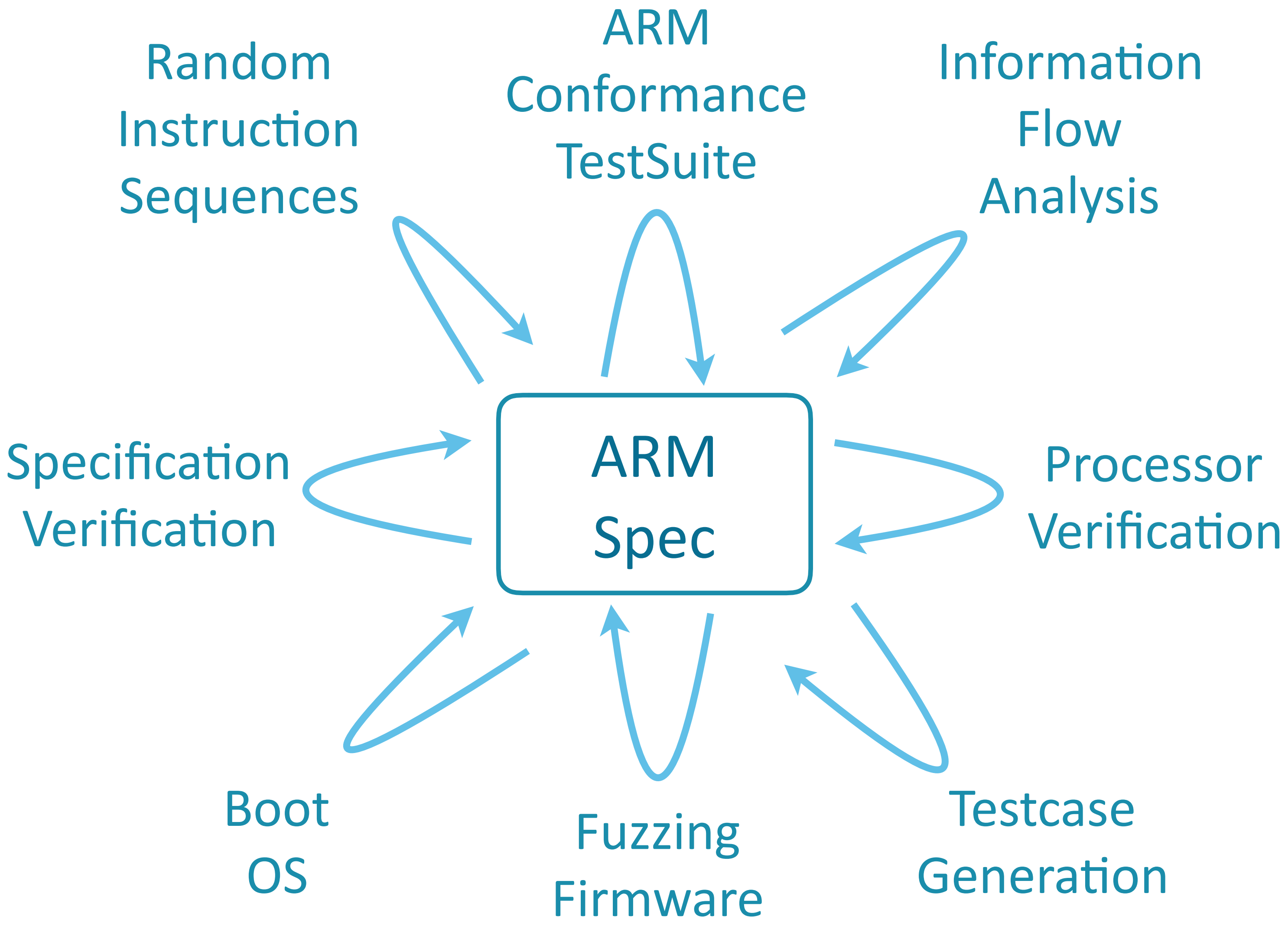
Measuring architecture coverage of tests

Untested: op1*op2 == -3.0, FPCR.RND=-Inf



```
                                    bits(N) FPRSqrtStepFused(bits(N) op1, bits(N) op2)
TESTED                                  assert N IN {32, 64};
TESTED                                  bits(N) result;
TESTED                                  op1 = FPNeg(op1);    // per FMSUB/FMLS
TESTED                                  (type1,sign1,value1) = FPUnpack(op1, FPCR);
TESTED                                  (type2,sign2,value2) = FPUnpack(op2, FPCR);
TESTED                                  (done,result) = FPProcessNaNs(type1, type2, op1, op2, FPCR);
TESTED      TESTED      TESTED          if !done then
TESTED                                      inf1 = (type1 == FPType_Infinity);
TESTED                                      inf2 = (type2 == FPType_Infinity);
TESTED                                      zero1 = (type1 == FPType_Zero);
TESTED                                      zero2 = (type2 == FPType_Zero);
TESTED      TESTED      TESTED              if (inf1 && zero2) || (zero1 && inf2) then
TESTED                                          result = FPOnePointFive('0');
                                            elsif inf1 || inf2 then
TESTED                                          result = FPInfinity(sign1 EOR sign2, N);
                                            else
                                                // Fully fused multiply-add and halve
TESTED                                          result_value = (3.0 + (value1 * value2)) / 2.0;
TESTED      UNEXECUTED  TESTED                  if result_value == 0.0 then
                                                    // Sign of exact zero result depends on rounding mode
UNEXECUTED                                          sign = if FPCRRounding() == FPRounding_NEGINF then '1' else '0';
UNEXECUTED                                          result = FPZero(sign, N);
                                                else
TESTED                                              result = FPRound(result_value, FPCRRounding());
TESTED                                  return result;
```

Creating a Virtuous Cycle



ARM Spec

Random Instruction Sequences

ARM Conformance TestSuite

Information Flow Analysis

Processor Verification

Testcase Generation

Fuzzing Firmware

Boot OS

Specification Verification

“End to End Verification of ARM processors with ISA Formal,” CAV 2016

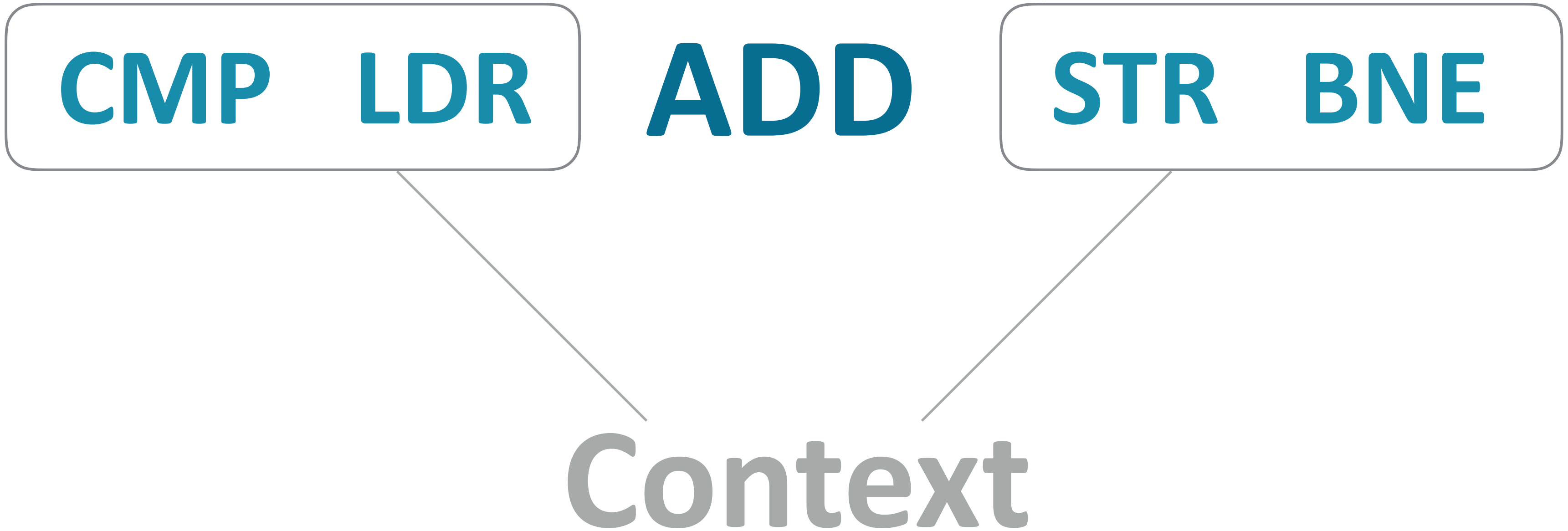
Formal validation of processors

arm

Checking an instruction

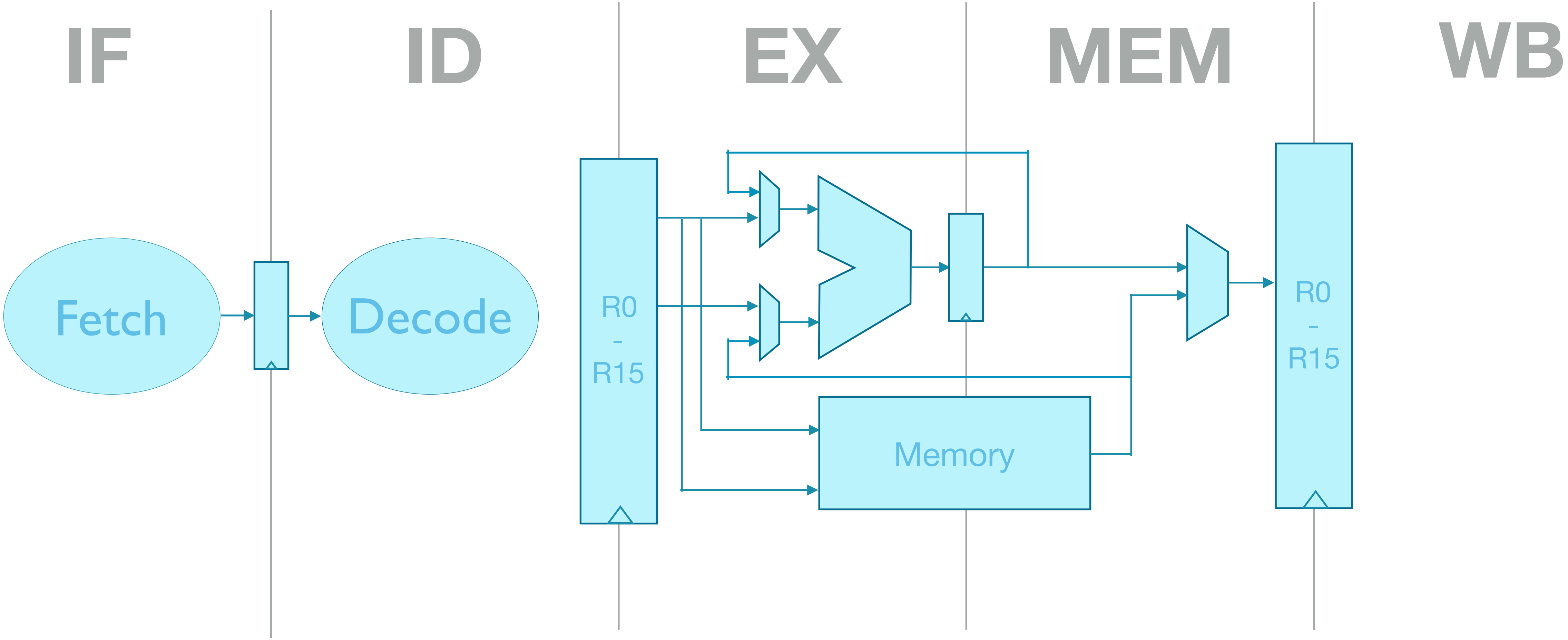
ADD

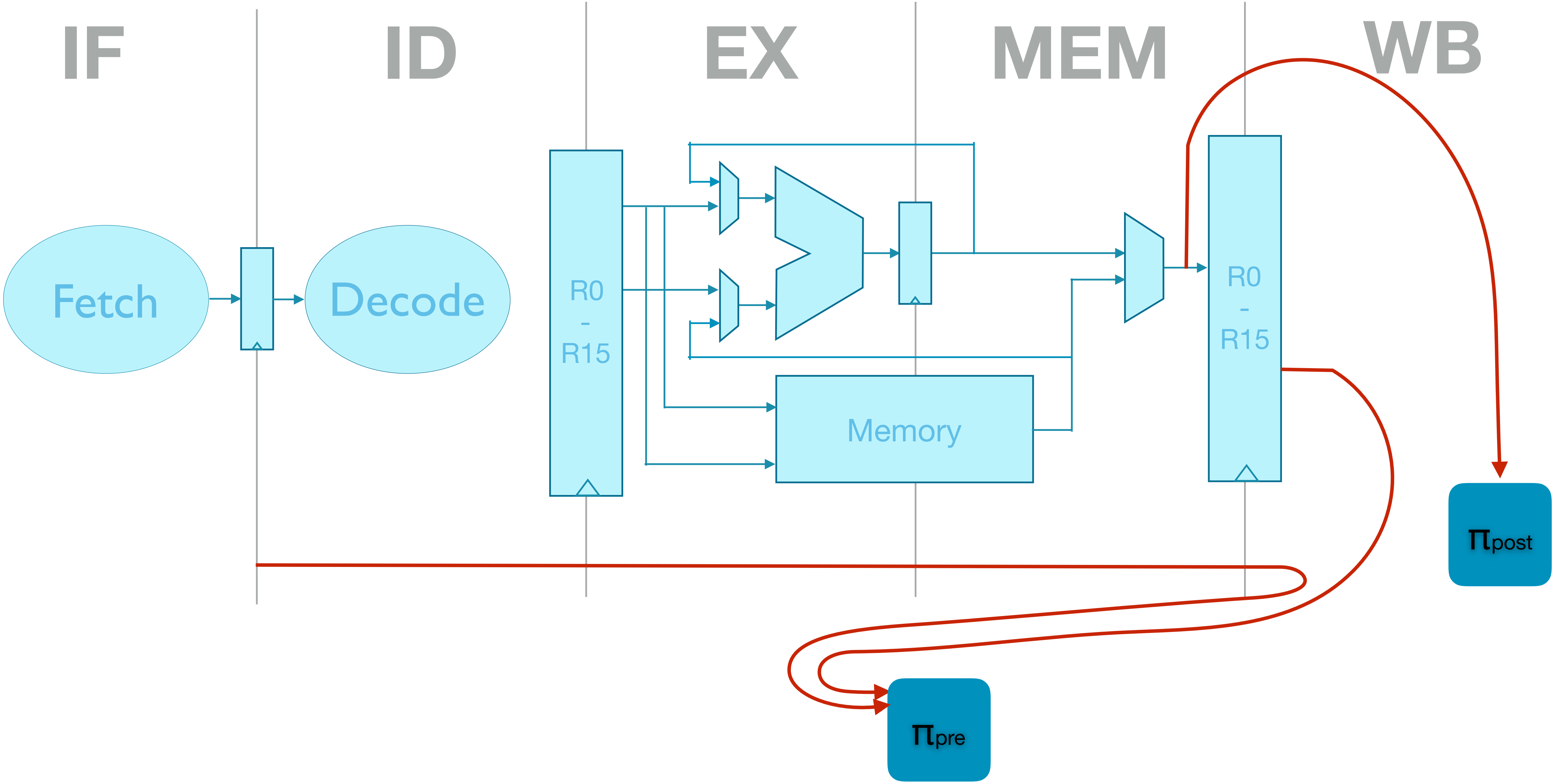
Checking an instruction

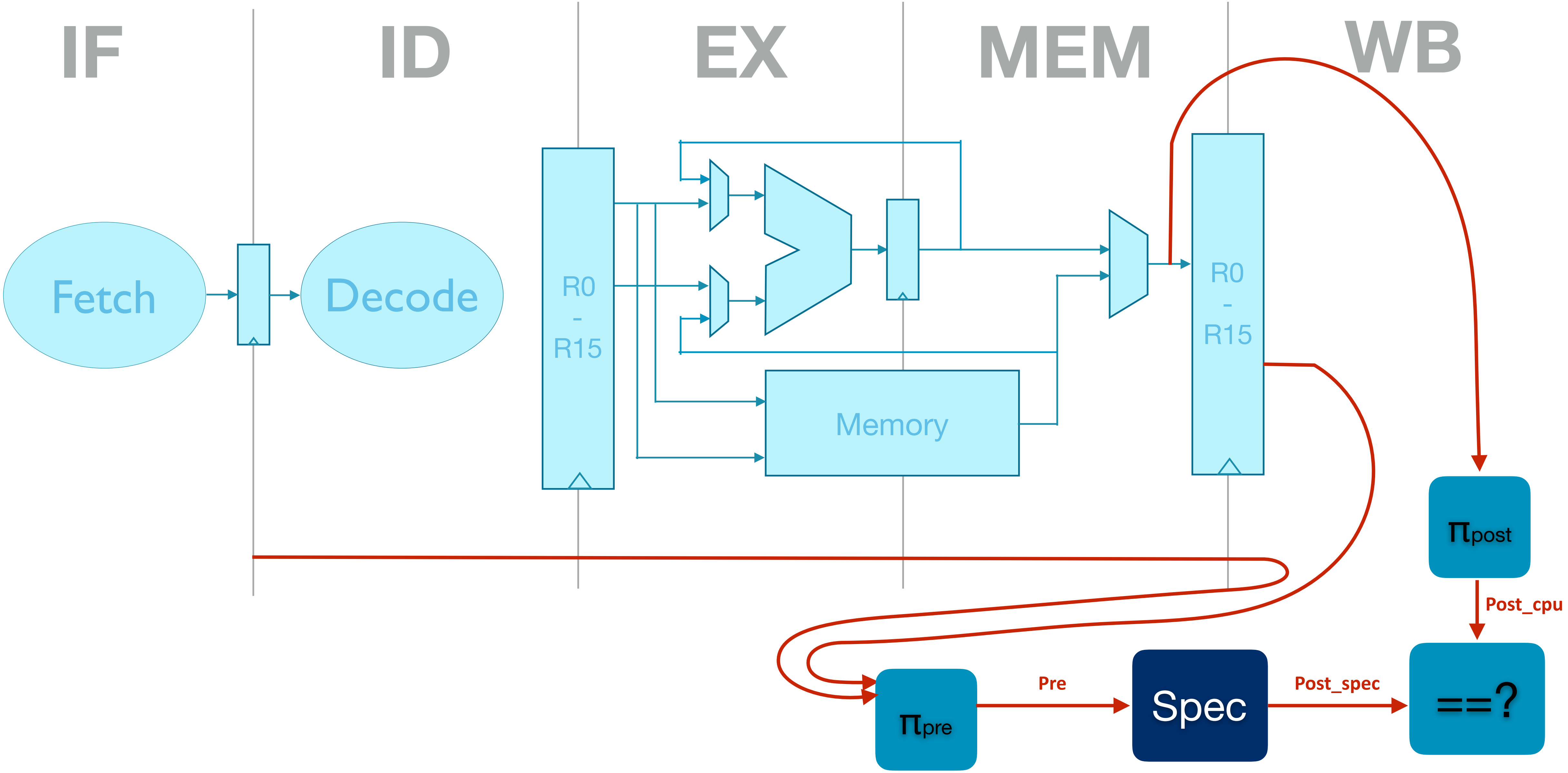


CMP LDR **ADD** STR BNE

Context


IF
ID
EX
MEM
WB
Fetch
Decode
R0
-
R15
Memory
R0
-
R15


IF
ID
EX
MEM
WB
Fetch
Decode
R0
-
R15
Memory
R0
-
R15
π_{post}
π_{pre}


IF
ID
EX
MEM
WB
Fetch
Decode
R0
-
R15
Memory
R0
-
R15
πpost
Post_cpu
πpre
Pre
Spec
Post_spec
==?

Errors ISA-Formal can catch

No Context

- Errors in decode
- Errors in data path

Context

- Errors in forwarding logic
- Errors in register renaming
- Errors in exception handling
- Errors in speculative execution

Specifying ADD

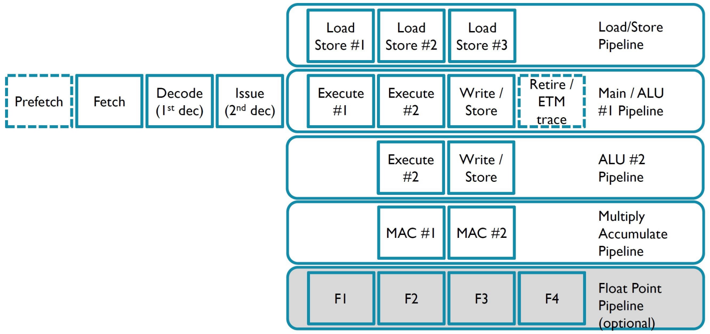
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	Rm			Rn			Rd		

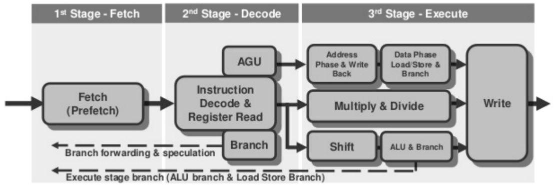
```
assign ADD_retiring = (pre.opcode & 16'b1111_1110_0000_0000)
                                  == 16'b0001_1000_0000_0000;
assign ADD_result   = pre.R[pre.opcode[8:6]] + pre.R[pre.opcode[5:3]];
assign ADD_Rd       = pre.opcode[2:0];


assert property (@(posedge clk) disable iff (~reset_n)
     ADD_retiring |-> (ADD_result == post.R[ADD_Rd]));
```

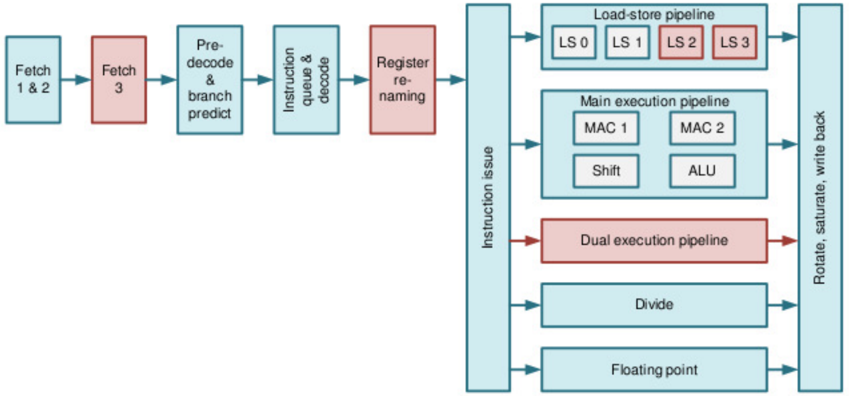
ISA Formal

- Finds complex bugs in processor pipelines
- Applied to wide range of µArchitectures
- Uses translation of ARM's internal ISA specification

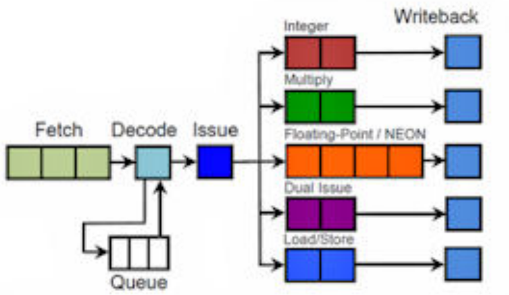

Prefetch
Fetch
Decode (1st dec)
Issue (2nd dec)
Load Store #1
Load Store #2
Load Store #3
Load/Store Pipeline
Execute #1
Execute #2
Write / Store
Retire / ETM trace
Main / ALU #1 Pipeline
Execute #2
Write / Store
ALU #2 Pipeline
MAC #1
MAC #2
Multiply Accumulate Pipeline
F1
F2
F3
F4
Float Point Pipeline (optional)



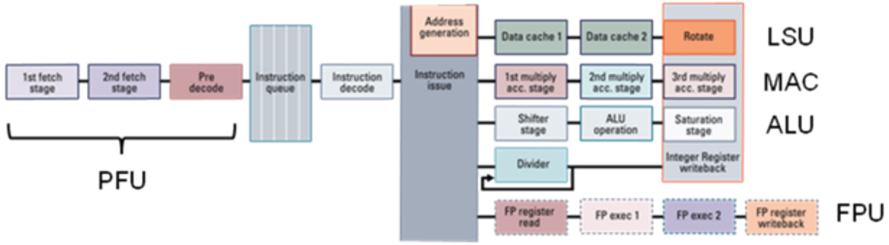
1st Stage - Fetch
2nd Stage - Decode
3rd Stage - Execute
Fetch (Prefetch)
AGU
Instruction Decode & Register Read
Branch
Address Phase & Write Back
Data Phase Load/Store & Branch
Multiply & Divide
Shift
ALU & Branch
Write
Branch forwarding & speculation
Execute stage branch (ALU branch & Load Store Branch)



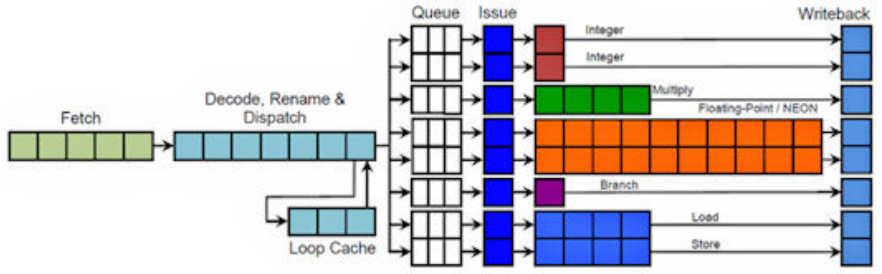
Fetch 1 & 2
Fetch 3
Pre-decode & branch predict
Instruction queue & decode
Register re-naming
Instruction issue
Load-store pipeline
LS 0
LS 1
LS 2
LS 3
Main execution pipeline
MAC 1
MAC 2
Shift
ALU
Dual execution pipeline
Divide
Floating point
Rotate, saturate, write back



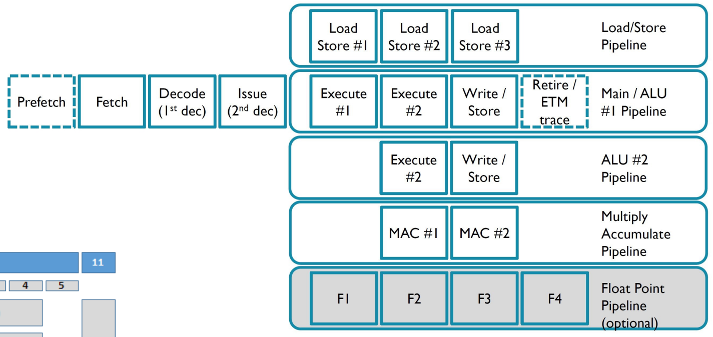
Fetch
Decode
Issue
Queue
Writeback
Integer
Multiply
Floating-Point / NEON
Dual Issue
Load/Store



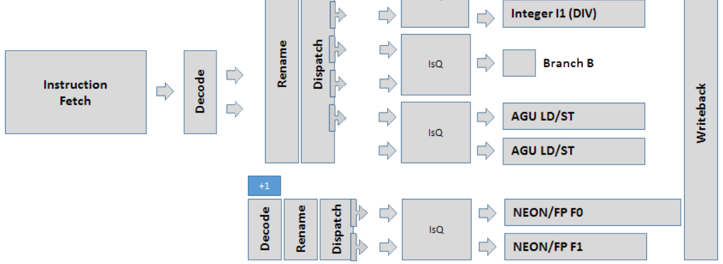
1st fetch stage
2nd fetch stage
Pre decode
PFU
Instruction queue
Instruction decode
Instruction issue
Address generation
Data cache 1
Data cache 2
Rotate
1st multiply acc. stage
2nd multiply acc. stage
3rd multiply acc. stage
Shifter stage
ALU operation
Saturation stage
Divider
Integer Register writeback
FP register read
FP exec 1
FP exec 2
FP register writeback
LSU
MAC
ALU
FPU



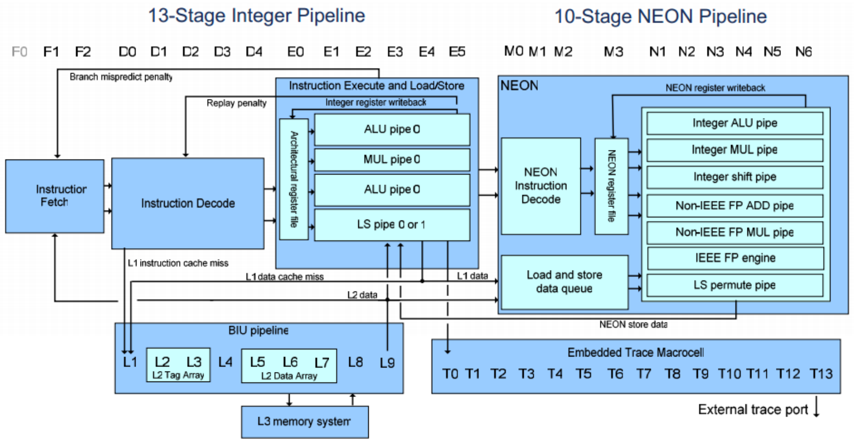
Queue
Issue
Writeback
Fetch
Decode, Rename & Dispatch
Loop Cache
Integer
Integer
Multiply
Floating-Point / NEON
Branch
Load
Store



Prefetch
Fetch
Decode (1st dec)
Issue (2nd dec)
Load Store #1
Load Store #2
Load Store #3
Load/Store Pipeline
Execute #1
Execute #2
Write / Store
Retire / ETM trace
Main / ALU #1 Pipeline
Execute #2
Write / Store
ALU #2 Pipeline
MAC #1
MAC #2
Multiply Accumulate Pipeline
F1
F2
F3
F4
Float Point Pipeline (optional)



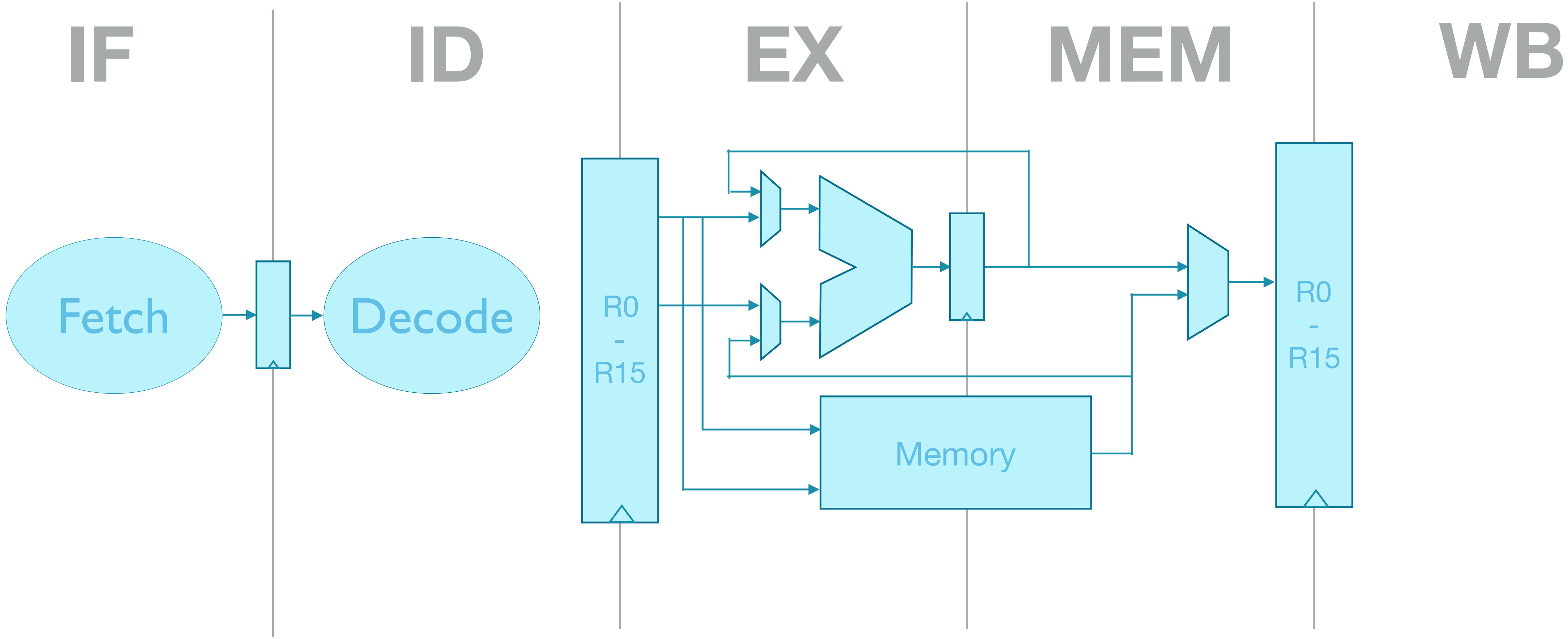
1
2
3
4
5
6
7
8
9
10
11
1
2
3
4
5
Instruction Fetch
Decode
Rename
Dispatch
IsQ
IsQ
IsQ
Integer I0 (MUL)
Integer I1 (DIV)
Branch B
AGU LD/ST
AGU LD/ST
+1
Decode
Rename
Dispatch
IsQ
NEON/FP F0
NEON/FP F1
Writeback

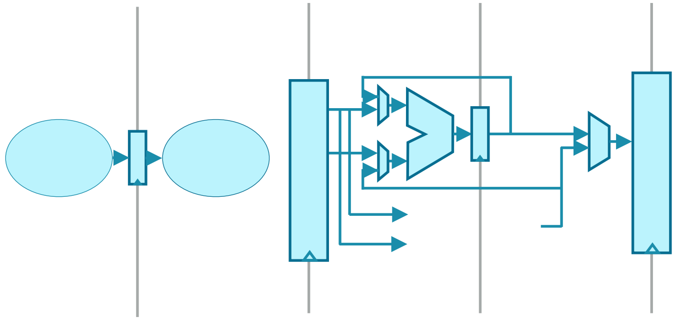


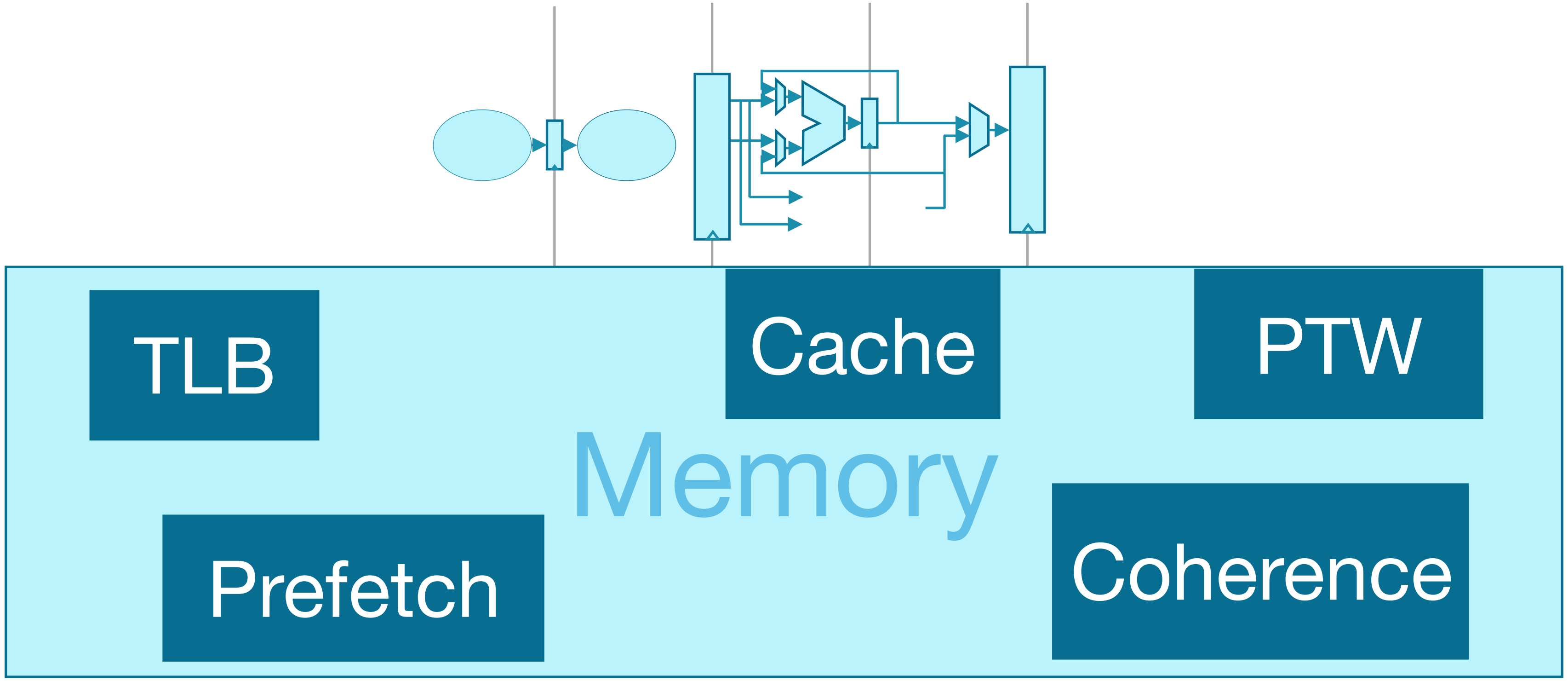
13-Stage Integer Pipeline
10-Stage NEON Pipeline
F0
F1
F2
D0
D1
D2
D3
D4
E0
E1
E2
E3
E4
E5
M0
M1
M2
M3
N1
N2
N3
N4
N5
N6
Branch mispredict penalty
Replay penalty
Instruction Execute and Load/Store
Integer register writeback
Instruction Fetch
Instruction Decode
Architectural register file
ALU pipe 0
MUL pipe 0
ALU pipe 0
LS pipe 0 or 1
L1 instruction cache miss
L1 data cache miss
L1 data
L2 data
NEON
NEON register writeback
NEON Instruction Decode
NEON register file
Integer ALU pipe
Integer MUL pipe
Integer shift pipe
Non-IEEE FP ADD pipe
Non-IEEE FP MUL pipe
IEEE FP engine
LS permute pipe
Load and store data queue
NEON store data
BIU pipeline
L1
L2
L3
L2 Tag Array
L4
L5
L6
L7
L2 Data Array
L8
L9
L3 memory system
Embedded Trace Macrocell
T0
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
T11
T12
T13
External trace port

Challenges

- Complex Functional Units
  - FP
  - Memory
- Dual Issue
- Instruction Fusion
- Register Renaming
- Out-of-order Retire


IF
ID
EX
MEM
WB
Fetch
Decode
R0
-
R15
Memory
R0
-
R15





TLB

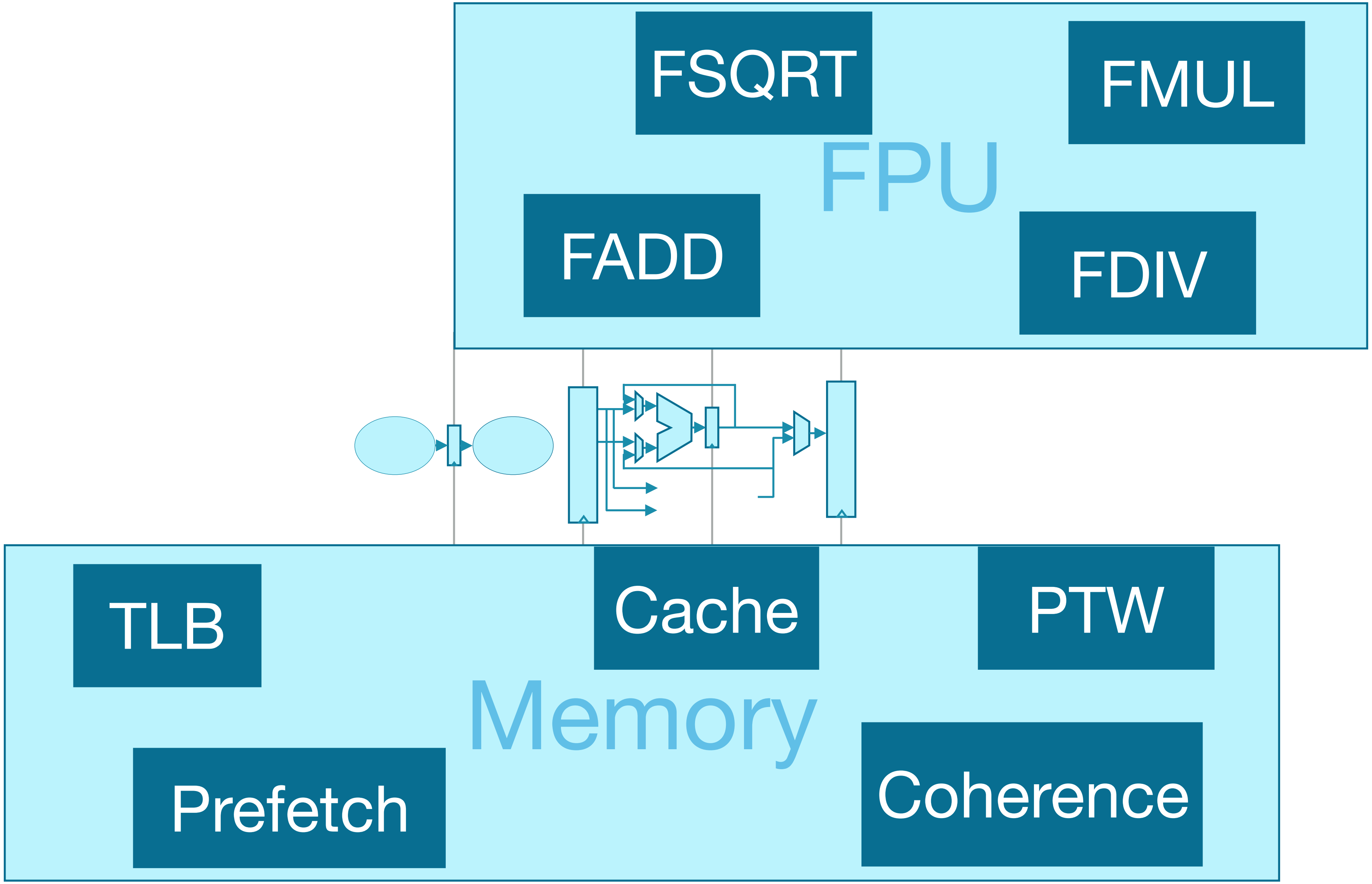
Cache

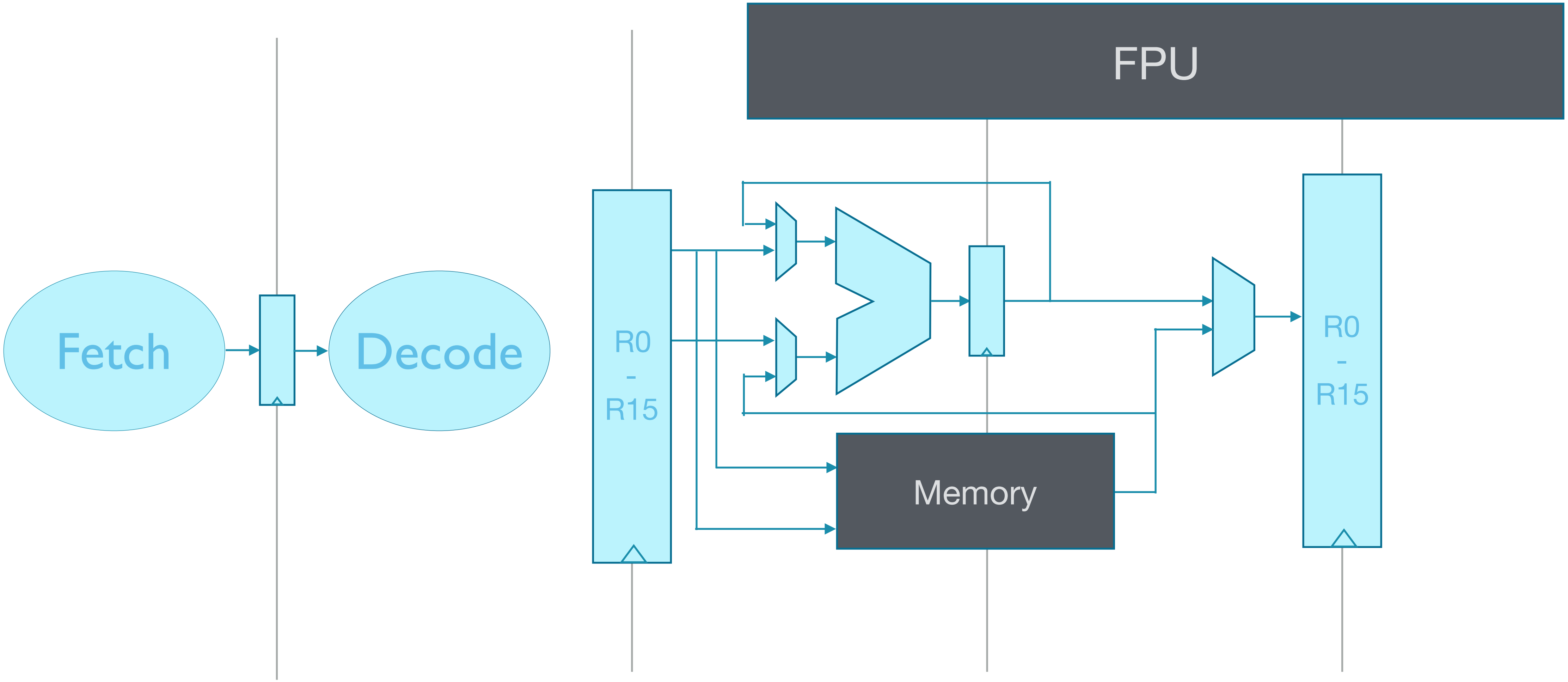
PTW

Memory

Prefetch

Coherence


FSQRT
FMUL
FPU
FADD
FDIV
TLB
Cache
PTW
Memory
Prefetch
Coherence


FPU
Fetch
Decode
R0
-
R15
Memory
R0
-
R15

FP Subset Behaviour

FPAdd	-∞	-1	0	1	∞
-∞	-∞	-∞	-∞	-∞	
-1	-∞		-1	0	∞
0	-∞	-1	0	1	∞
1	-∞	0	1		∞
∞		∞	∞	∞	∞

ISA Formal

- Finds complex bugs in processor pipelines
- Applied to wide range of µArchitectures
- Uses translation of ARM's internal ISA specification

ISA-Formal Properties

	ADC	ADD	B	…	YIELD
R[]		✓			
NZCV					
SP					
PC					
S[],D[],V[]					
FPSR					
MemRead					
MemWrite					
SysRegRW					
ELR					
ESR					
…					

ISA-Formal Properties

	ADC	ADD	B	…	YIELD
R[]		✓			
NZCV					
SP		✓			
PC					
S[],D[],V[]					
FPSR					
MemRead					
MemWrite					
SysRegRW					
ELR					
ESR					
…					

ISA-Formal Properties

	ADC	ADD	B	…	YIELD
R[]		✓	✓		
NZCV					
SP		✓			
PC			✓		
S[],D[],V[]					
FPSR					
MemRead					
MemWrite					
SysRegRW					
ELR					
ESR					
…					

ISA-Formal Properties

	ADC	ADD	B	…	YIELD
R[]	✓	✓	✓		
NZCV	✓				
SP	✓	✓			
PC			✓		
S[],D[],V[]					
FPSR					
MemRead					
MemWrite					
SysRegRW					
ELR					
ESR					
…					

But this is slow and inconsistent

ISA-Formal Properties

	ADC	ADD	B	…	YIELD
R[]	✔	✔	✔	✔	✔
NZCV					
SP					
PC					
S[],D[],V[]					
FPSR					
MemRead					
MemWrite					
SysRegRW					
ELR					
ESR					
…					

ISA-Formal Properties

	ADC	ADD	B	…	YIELD
R[]	✔	✔	✔	✔	✔
NZCV	✔	✔	✔	✔	✔
SP	✔	✔	✔	✔	✔
PC	✔	✔	✔	✔	✔
S[],D[],V[]					
FPSR					
MemRead					
MemWrite					
SysRegRW					
ELR					
ESR					
…					

ISA-Formal Properties

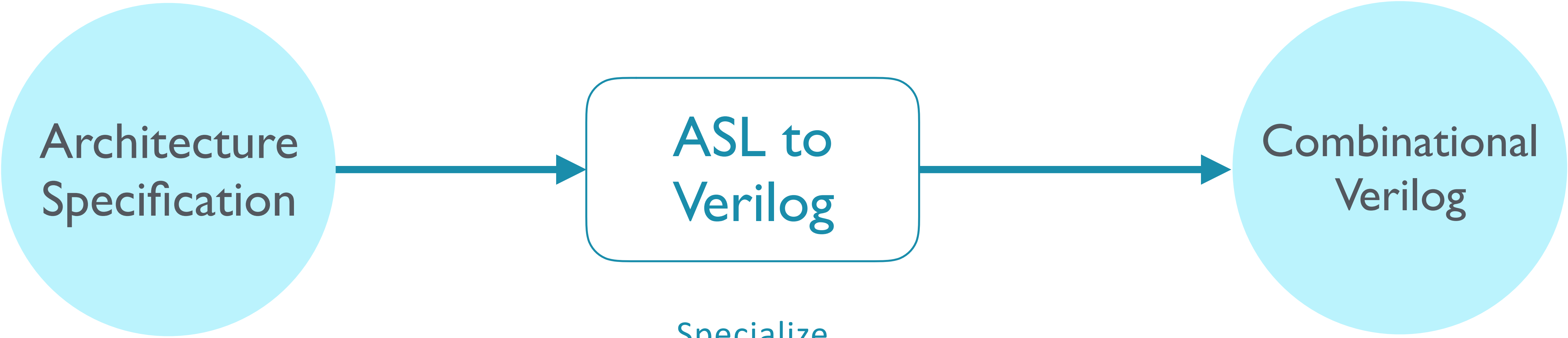
	ADC	ADD	B	...	YIELD
R[]	✔	✔	✔	✔	✔
NZCV	✔	✔	✔	✔	✔
SP	✔	✔	✔	✔	✔
PC	✔	✔	✔	✔	✔
S[],D[],V[]	✔	✔	✔	✔	✔
FPSR	✔	✔	✔	✔	✔
MemRead					
MemWrite					
SysRegRW					
ELR					
ESR					
...					

ISA-Formal Properties

	ADC	ADD	B	...	YIELD
R[]	✔	✔	✔	✔	✔
NZCV	✔	✔	✔	✔	✔
SP	✔	✔	✔	✔	✔
PC	✔	✔	✔	✔	✔
S[],D[],V[]	✔	✔	✔	✔	✔
FPSR	✔	✔	✔	✔	✔
MemRead	✔	✔	✔	✔	✔
MemWrite	✔	✔	✔	✔	✔
SysRegRW					
ELR					
ESR					
...					

ISA-Formal Properties

	ADC	ADD	B	...	YIELD
R[]	✓	✓	✓	✓	✓
NZCV	✓	✓	✓	✓	✓
SP	✓	✓	✓	✓	✓
PC	✓	✓	✓	✓	✓
S[],D[],V[]	✓	✓	✓	✓	✓
FPSR	✓	✓	✓	✓	✓
MemRead	✓	✓	✓	✓	✓
MemWrite	✓	✓	✓	✓	✓
SysRegRW	✓	✓	✓	✓	✓
ELR	✓	✓	✓	✓	✓
ESR	✓	✓	✓	✓	✓
...					

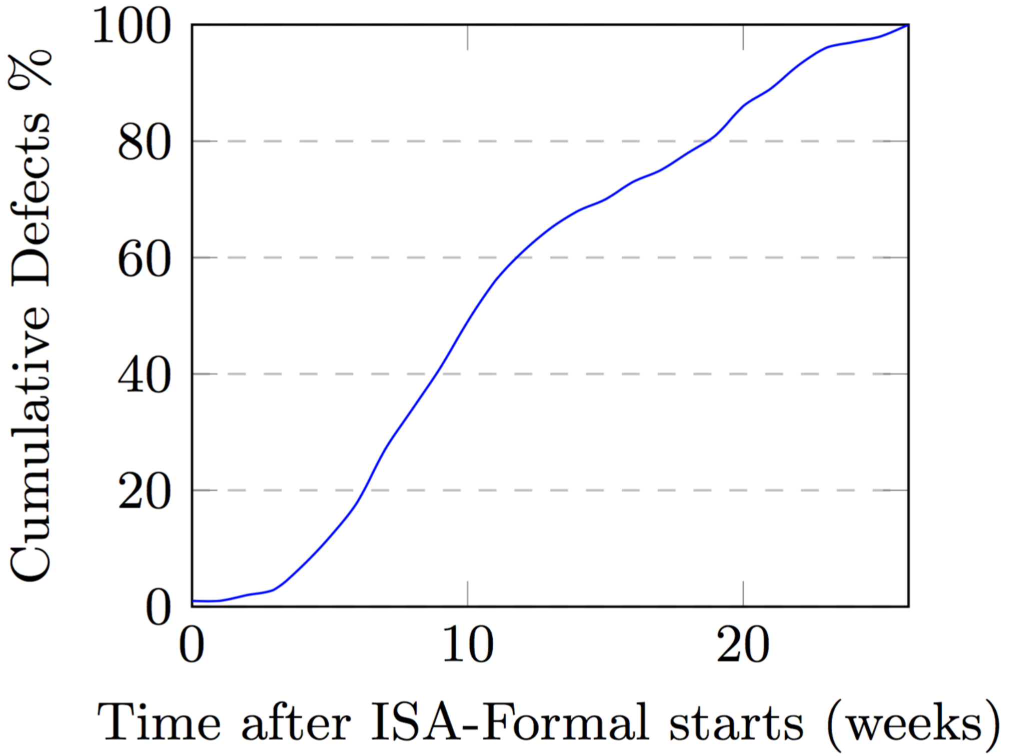


Architecture Specification

ASL to Verilog

Combinational Verilog

Specialize
Monomorphize
Constant Propagation
Width Analysis
Exception Handling
…



FP/SIMD	25%
Memory	21%
Branch	21%
Integer	18%
Exception	8%
System	7%

Defect Detection by Area

Arm CPUs verified with ISA-Formal

A-class

Cortex-A53

Cortex-A32

Cortex-A35

Cortex-A55

Next generation

R-class

Cortex-R52

Next generation

M-class

Cortex-M4

Cortex-M7

Cortex-M33

Next generation

Cambridge Projects

Rolling out globally to other design centres

Sophia, France - Cortex-A75 (partial)

Austin, USA - TBA

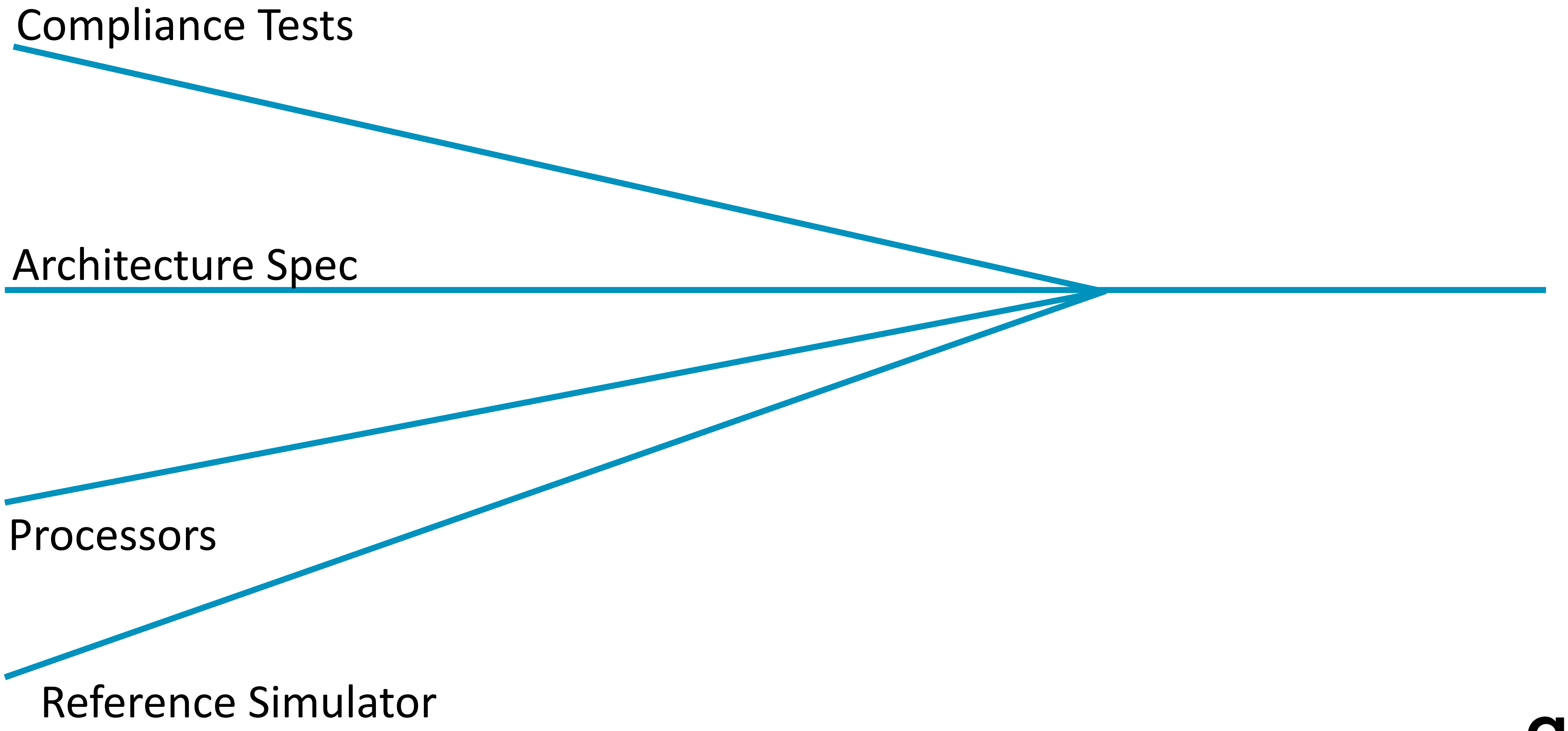
Chandler, USA - TBA

“Who guards the guards? Formal Validation of ARM v8-M Specifications” OOPSLA 2017

Formal validation of specifications

arm

One Specification to rule them all?



Creating a redundant specification

Where to get a list of redundant properties from?

How to formalise this list?

How to formally validate specification against properties?

(This may look familiar from formal specification of software)

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 is by any of the following:

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

Rule R

State Change X is by any of the following:

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

And cannot happen any other way

Rule R

State Change X is by any of the following:

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

And cannot happen any other way

$$\text{Rule R:} \quad X \rightarrow A \vee B \vee C \vee D$$

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)

$$\begin{aligned}\mathrm{Fell}(\mathrm{LockedUp}) \rightarrow\ & \mathrm{Called}(\mathrm{TakeColdReset}) \\ \lor\ & \mathrm{Called}(\mathrm{TakeReset}) \\ \lor\ & \mathrm{Rose}(\mathrm{Halted}) \\ \lor\ & \mathrm{Called}(\mathrm{ExceptionEntry})\end{aligned}$$

Rule VGNW

Entry to lockup from an exception causes

- Any Fault Status Registers associated with the exception to be updated.

Out of date

- No update to the exception state, pending or active.

Misleading

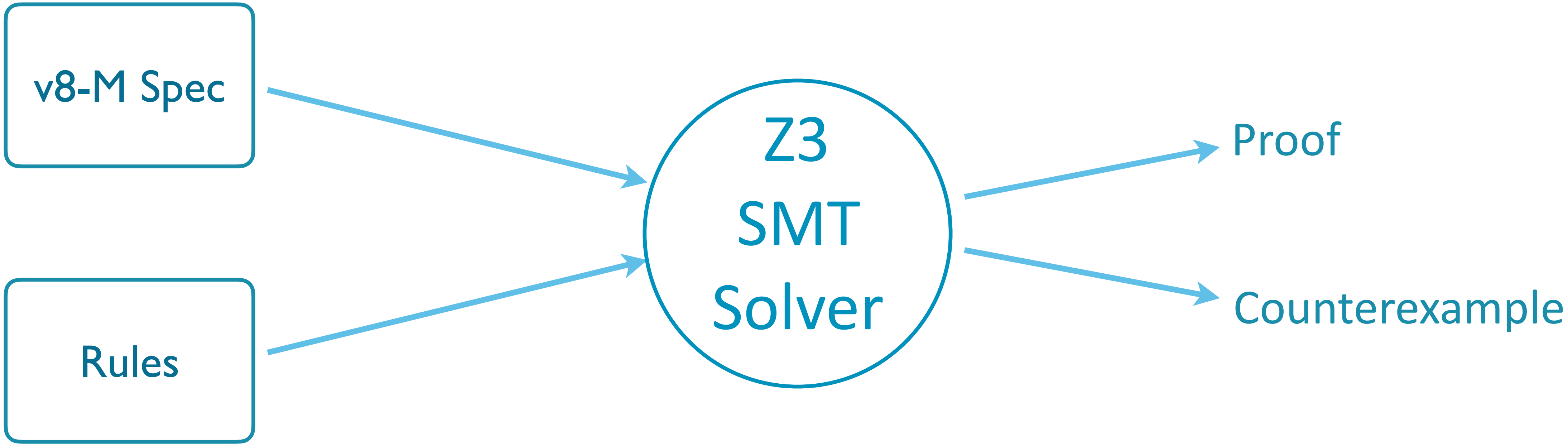
- The PC to be set to 0xEFFFFFFE.

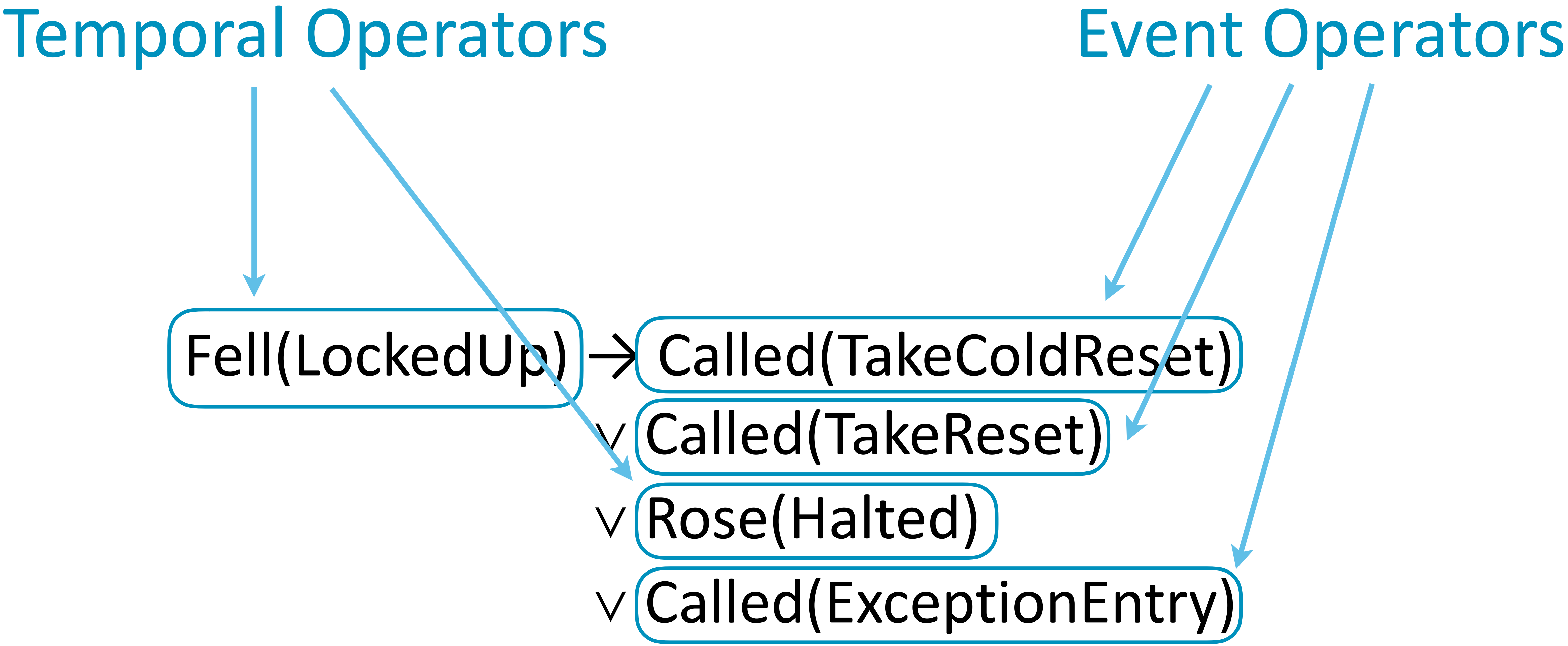
Untestable

- EPSR.IT to become UNKNOWN.

Ambiguous

In addition, HFSR.FORCED is not set to 1.


v8-M Spec
Rules
Z3
SMT
Solver
Proof
Counterexample

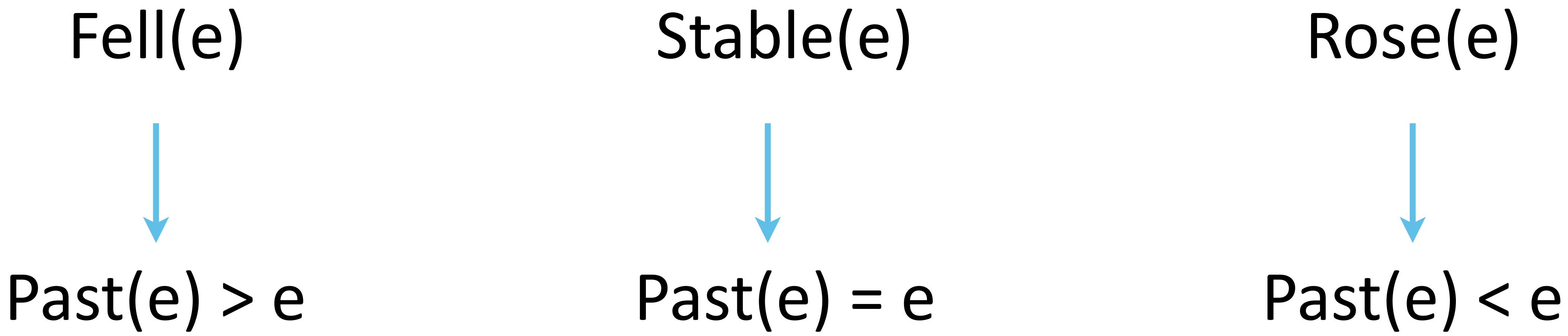


Temporal Operators

Event Operators

Fell(LockedUp) → Called(TakeColdReset)
∨ Called(TakeReset)
∨ Rose(Halted)
∨ Called(ExceptionEntry)

Temporal Operators



Fell(e)

Past(e) > e

Stable(e)

Past(e) = e

Rose(e)

Past(e) < e

Temporal Operators

Fell(LockedUp)

```
__Past_LockedUp = LockedUp;

FunctionUnderTest();

… __Past_LockedUp > LockedUp …
```

Event Operators

Called(TakeReset)

```
TakeReset()
{
	__Called_TakeReset = TRUE;
	…
}
```

```
__Called_TakeColdReset      = FALSE;
__Called_TakeReset          = FALSE;
__Called_TakeExceptionEntry = FALSE;
__Past_LockedUp = LockedUp;
__Past_Halted   = Halted;

FunctionUnderTest();

assert((__Past_LockedUp > LockedUp)
       ==>
       (  __Called_TakeColdReset
       || __Called_TakeReset
       || __Past_Halted < Halted
       || __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.

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

```
__Called_TakeColdReset      = FALSE;    assert((__Past_LockedUp > LockedUp)
__Called_TakeReset          = FALSE;           ==>
__Called_TakeExceptionEntry = FALSE;           (  __Called_TakeColdReset
__Past_LockedUp = LockedUp;                    || __Called_TakeReset
__Past_Halted   = Halted;                      || __Past_Halted < Halted
                                               || __Called_ExceptionEntry));
```

Arm Specification Language → SMT

Arithmetic operations	Arithmetic operations
Boolean operations	Boolean operations
Bit Vectors	Bit Vectors
Arrays	Arrays
Functions	~~Functions~~
Local Variables	~~Local Variables~~
Statements	~~Statements~~
Assignments	~~Assignments~~
If-statements	~~If-statements~~
Loops	~~Loops~~
Exceptions	~~Exceptions~~

Results (more in OOPSLA paper)

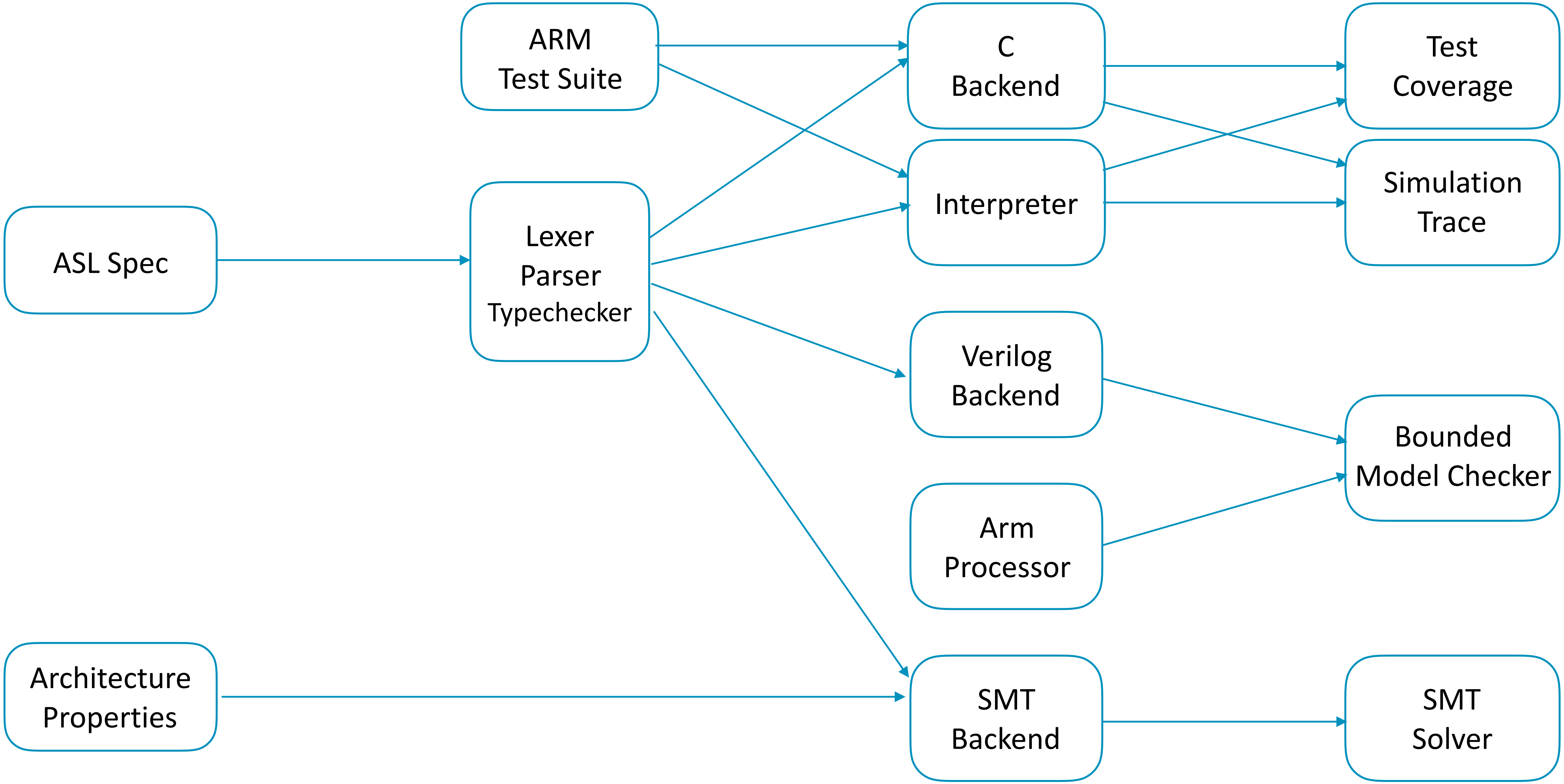
Most properties proved in under 100 seconds

Found 12 bugs in specification:

- debug, exceptions, system registers, security

Found bugs in English prose:

- ambiguous, imprecise, incorrect, …


ARM Test Suite
ASL Spec
Lexer Parser Typechecker
Architecture Properties
C Backend
Interpreter
Verilog Backend
Arm Processor
SMT Backend
Test Coverage
Simulation Trace
Bounded Model Checker
SMT Solver

Public release of machine readable Arm specification

Enable formal verification of software and tools

Releases

- April 2017: v8.2
- July 2017: v8.3

Working with Cambridge University REMS group to convert to SAIL

- Backends for HOL, OCaml, Memory model, (hopefully Coq too)

Specification: https://developer.arm.com/products/architecture/a-profile/exploration-tools

Tools: https://github.com/alastairreid/mra_tools

(See also: https://github.com/herd/herdtools7/blob/master/herd/libdir/aarch64.cat)

Talk to me about how I can help you use it

Specifications: The next bottleneck

We will need a lot of specs

- Of real world s/w + h/w

Specs are a large part of TCB

How are we going to create them?

How are we going to trust them?

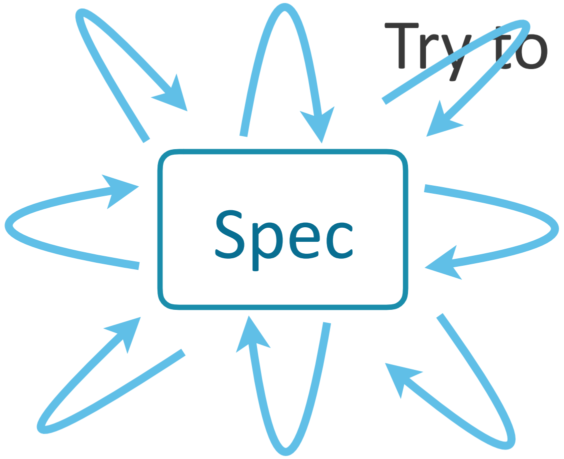
Test the specifications you depend on

Formally validate/verify implementations

Create redundant specifications

Ensure specifications have many uses

- Don’t write spec in Coq/HOL/ACL2/…
- Try to influence official specification



Thanks

Alasdair Armstrong (Cambridge U.)
Alex Chadwick (ARM)
Ali Zaidi (ARM)
Anastasios Deligiannis (ARM)
Anthony Fox (Cambridge U.)
Ashan Pathirane (ARM)
Belaji Venu (ARM)
Bradley Smith (ARM)
Brian Foley (ARM)
Curtis Dunham (ARM)
David Gilday (ARM)
David Hoyes (ARM)
David Seal (ARM)
Daniel Bailey (ARM)
Erin Shepherd (ARM)
Francois Botman (ARM)
George Hawes (ARM)
Graeme Barnes (ARM)
Isobel Hooper (ARM)
Jack Andrews (ARM)
Jacob Eapen (ARM)
Jon French (Cambridge U.)
Kathy Gray (Cambridge U.)
Krassy Gochev (ARM)
Lewis Russell (ARM)
Matthew Leach (ARM)
Meenu Gupta (ARM)
Michele Riga (ARM)
Milosch Meriac (ARM)
Nigel Stephens (ARM)
Niyas Sait (ARM)
Peng Wang (ARM)
Peter Sewell (Cambridge U.)
Peter Vrabel (ARM)
Richard Grisenthwaite (ARM)
Rick Chen (ARM)
Simon Bellew (ARM)
Thomas Grocutt (ARM)
Will Deacon (ARM)
Will Keen (ARM)
Wojciech Meyer (ARM)
(and others)

Thank You!

Danke!

Merci!

谢谢!

ありがとう!

Gracias!

Kiitos!

@alastair_d_reid

arm

“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