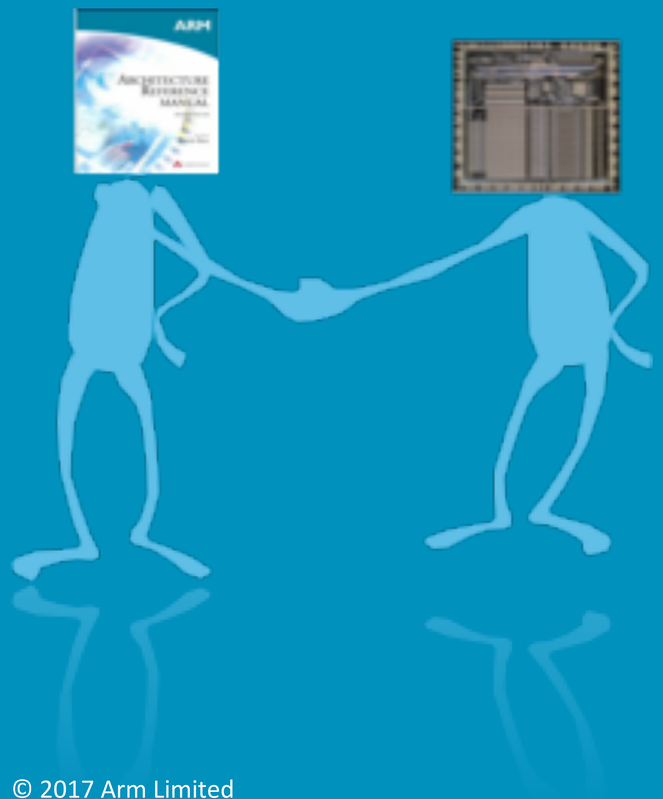
arm

34th Chaos Communication Congress

34C3: bUWaB!

How Can You Trust Formally Verified Software?



Alastair Reid

@alastair_d_reid

ARM Research

Arm Processor Architecture

Widely used in many different areas: phones, tablets, IoT, HDD, …

Important to understand what they do

Important to be able to analyse malware, security analysis, etc.

April 2011: Started work on formal specifications of ARM processor architectures

April 2017: Public release in machine readable form

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

Working with REMS @ Cambridge Uni to translate ARM spec to SAIL to HOL/OCaml/…

arm

What can you do with an executable processor specification

How can you trust formally verified software?

arm

ARM Machine Readable Architecture Specification

Instructions

Security features: memory protection, exceptions, privilege checks, TrustZone, …

Links

- Official ARM release https://developer.arm.com/products/architecture/a-profile/exploration-tools
- HTML files (part of official release) https://www.meriac.com/archex/
- Tools to dissect the official release (incl. parser) https://github.com/alastairreid/mra_tools
- Blog article about release https://alastairreid.github.io/ARM-v8a-xml-release/
- Papers
  - "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

arm

https://www.meriac.com/archex/ARMv83A-SysReg/AArch32-sctlr.xml

SCTLR, System Control Register

The SCTLR characteristics are:

Purpose

Provides the top level control of the system, including its memory system.

arm

https://www.meriac.com/archex/ARMv83A-SysReg/AArch32-sctlr.xml

SCTLR, System Control Register

The SCTLR characteristics are:

Purpose

Provides the top level control of the system, including its memory system.

Field descriptions

The SCTLR bit assignments are:

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
0	TE	AFE	TRE	0	0	EE	0	SPAN	1	0	UWXN	WXN	nTWE	0	nTWI	0	0	V	I	1	0	0	SED	ITD	UNK	CP15BEN	LSMAOE	nTLSMD	C	A	M

arm

https://www.meriac.com/archex/ARMv83A-SysReg/AArch32-sctlr.xml

SCTLR, System Control Register

The SCTLR characteristics are:

Purpose

Provides the top level control of the system, including its memory system.

Field descriptions

The SCTLR bit assignments are:

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
0	TE	AFE	TRE	0	0	EE	0	SPAN	1	0	UWXN	WXN	nTWE	0	nTWI	0	0	V	I	1	0	0	SED	ITD	UNK	CP15BEN	LSMAOE	nTLSMD	C	A	M

WXN, bit [19]

Write permission implies XN (Execute-never). For the PL1&0 translation regime, this bit can force all memory regions that are writable to be treated as XN. The possible values of this bit are:

WXN	Meaning
0	This control has no effect on memory access permissions.
1	Any region that is writable in the PL1&0 translation regime is forced to XN for accesses from software executing at PL1 or PL0.

The WXN bit is permitted to be cached in a TLB.

When this register has an architecturally-defined reset value, this field resets to 0.

arm

```
MRC p15, 0, R0, c1, c0, 0
ORR R0, R0, #0x80000
MCR p15, 0, R0, c1, c0, 0
```


See also: https://github.com/gdelugre/ida-arm-system-highlight

arm

```
MRC p15, 0, R0, c1, c0, 0
ORR R0, R0, #0x80000
MCR p15, 0, R0, c1, c0, 0

MRC R0, SCTLR
ORR R0, R0, #0x80000
MCR R0, SCTLR
```


See also: https://github.com/gdelugre/ida-arm-system-highlight

```
MRC p15, 0, R0, c1, c0, 0
ORR R0, R0, #0x80000
MCR p15, 0, R0, c1, c0, 0

MRC R0, SCTLR
ORR R0, R0, #0x80000
MCR R0, SCTLR

SCTLR.WXN = 1;
```


See also: https://github.com/gdelugre/ida-arm-system-highlight

```
MRC p15, 0, R0, c1, c0, 0
ORR R0, R0, #0x80000
MCR p15, 0, R0, c1, c0, 0

MRC R0, SCTLR
ORR R0, R0, #0x80000
MCR R0, SCTLR

SCTLR.WXN = 1;
```

Write permission implies XN (Execute-never). For the PL1&0 translation regime, this bit can force all memory regions that are writable to be treated as XN. The possible values of this bit are:

WXN	Meaning
0	This control has no effect on memory access permissions.
1	Any region that is writable in the PL1&0 translation regime is forced to XN for accesses from software executing at PL1 or PL0.

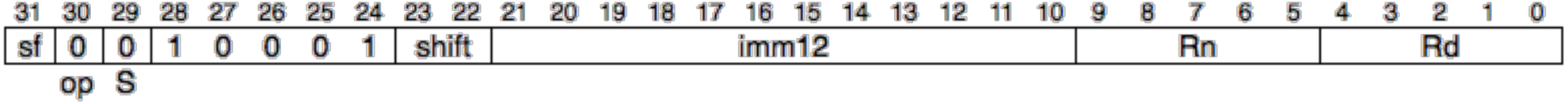
See also: https://github.com/gdelugre/ida-arm-system-highlight

https://www.meriac.com/archex/A64_v83A_ISA/add_addsub_imm.xml

ADD (immediate)

Add (immediate) adds a register value and an optionally-shifted immediate value, and writes the result to the destination register.

This instruction is used by the alias MOV (to/from SP).



32-bit (sf == 0)

```
ADD <Wd|WSP>, <Wn|WSP>, #<imm>{, <shift>}
```

64-bit (sf == 1)

```
ADD <Xd|SP>, <Xn|SP>, #<imm>{, <shift>}
```

```
integer d = UInt(Rd);
integer n = UInt(Rn);
integer datasize = if sf == '1' then 64 else 32;
bits(datasize) imm;

case shift of
   when '00' imm = ZeroExtend(imm12, datasize);
   when '01' imm = ZeroExtend(imm12:Zeros(12), datasize);
   when '1x' ReservedValue();
```

Assembler Symbols

<Wd|WSP>	Is the 32-bit name of the destination general-purpose register or stack pointer, encoded in the "Rd" field.
<Wn|WSP>	Is the 32-bit name of the source general-purpose register or stack pointer, encoded in the "Rn" field.
<Xd|SP>	Is the 64-bit name of the destination general-purpose register or stack pointer, encoded in the "Rd" field.
<Xn|SP>	Is the 64-bit name of the source general-purpose register or stack pointer, encoded in the "Rn" field.
<imm>	Is an unsigned immediate, in the range 0 to 4095, encoded in the "imm12" field.
<shift>	Is the optional left shift to apply to the immediate, defaulting to LSL #0 and encoded in "shift":

shift	<shift>
00	LSL #0
01	LSL #12
1x	RESERVED

Alias Conditions

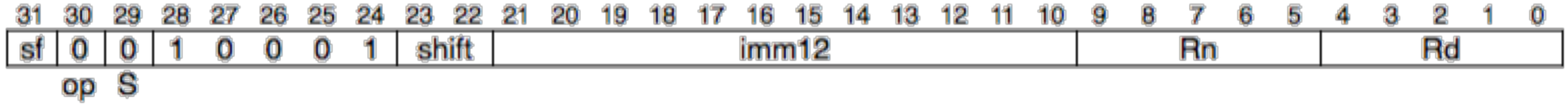
Alias	Is preferred when
MOV (to/from SP)	shift == '00' && imm12 == '000000000000' && (Rd == '11111' || Rn == '11111')

Operation

```
bits(datasize) result;
bits(datasize) operand1 = if n == 31 then SP[] else X[n];

(result, -) = AddWithCarry(operand1, imm, '0');

if d == 31 then
   SP[] = result;
else
   X[d] = result;
```

Assembler / Disassembler

https://alastairreid.github.io/bidirectional-assemblers/




```
ADD <Xd|SP>, <Xn|SP>, #<imm>{, <shift>}
```

```
[sf:"1"; op:"0"; S:"0"; "10001"; shift:"xx"; imm12:"xxxxxxxxxxxx"; Rn:"xxxxx"; Rd:"xxxxx"]
<->
"ADD" " " <Xd|SP> "," " " <Xn|SP> "," " " [ "#" ] <imm> " " [ "," " " <shift> ]
  where
    <Xd|SP> = RegXSP(UInt(Rd));
    <Xn|SP> = RegXSP(UInt(Rn));
    <imm>   = UInt(imm12);
    <shift> = Optional("LSL #0",
                case shift {
                 '00' <-> "LSL #0";
                 '01' <-> "LSL #12";
                 '1x' <-> RESERVED();
                });
```


arm

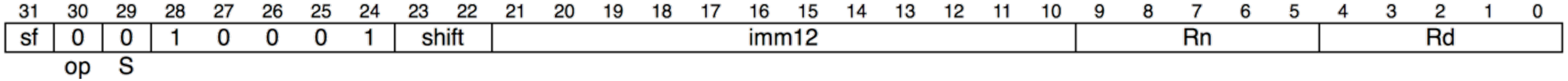
See also: https://github.com/agustingianni/retools and https://github.com/nspin/hs-arm

ADD (immediate)

https://www.meriac.com/archex/A64_v83A_ISA/add_addsub_imm.xml

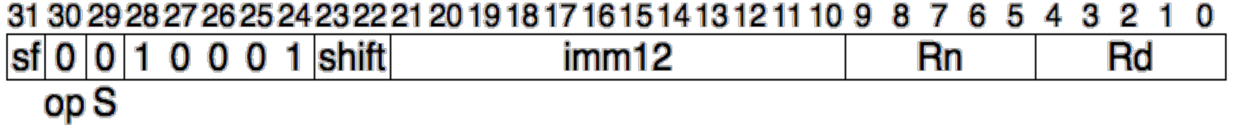
Add (immediate) adds a register value and an optionally-shifted immediate value, and writes the result to the destination register.

This instruction is used by the alias MOV (to/from SP).

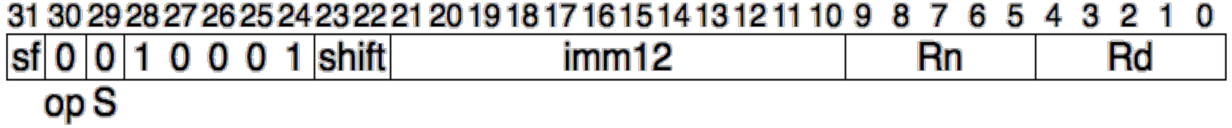



```
integer d = UInt(Rd);
integer n = UInt(Rn);
integer datasize = if sf == '1' then 64 else 32;
bits(datasize) imm;

case shift of
    when '00' imm = ZeroExtend(imm12, datasize);
    when '01' imm = ZeroExtend(imm12:Zeros(12), datasize);
    when '1x' ReservedValue();
bits(datasize) result;
bits(datasize) operand1 = if n == 31 then SP[] else X[n];

(result, -) = AddWithCarry(operand1, imm, '0');

if d == 31 then
    SP[] = result;
else
    X[d] = result;
```


arm




```
integer d = UInt(Rd);
integer n = UInt(Rn);
integer datasize = if sf == '1' then 64 else 32;
bits(datasize) imm;

case shift of
    when '00' imm = ZeroExtend(imm12, datasize);
    when '01' imm = ZeroExtend(imm12:Zeros(12), datasize);
    when '1x' ReservedValue();
bits(datasize) result;
bits(datasize) operand1 = if n == 31 then SP[] else X[n];

(result, -) = AddWithCarry(operand1, imm, '0');

if d == 31 then
    SP[] = result;
else
    X[d] = result;
```



sf = ‘0’

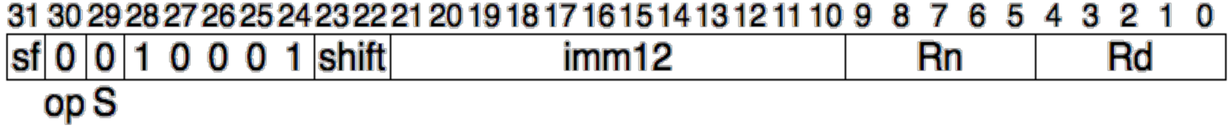
imm12 = 0x02a

shift = ‘01’

Rd = ‘00101’

Rn = ’00011’

```
integer d = UInt(Rd);
integer n = UInt(Rn);
integer datasize = if sf == '1' then 64 else 32;
bits(datasize) imm;

case shift of
    when '00' imm = ZeroExtend(imm12, datasize);
    when '01' imm = ZeroExtend(imm12:Zeros(12), datasize);
    when '1x' ReservedValue();
bits(datasize) result;
bits(datasize) operand1 = if n == 31 then SP[] else X[n];

(result, -) = AddWithCarry(operand1, imm, '0');

if d == 31 then
    SP[] = result;
else
    X[d] = result;
```

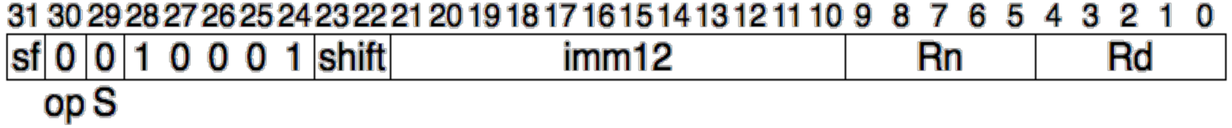


sf	=	‘0’
imm12	=	0x02a
shift	=	‘01’
Rd	=	‘00101’
Rn	=	’00011’

```
integer d = UInt(Rd);
integer n = UInt(Rn);
integer datasize = if sf == '1' then 64 else 32;
bits(datasize) imm;

case shift of
    when '00' imm = ZeroExtend(imm12, datasize);
    when '01' imm = ZeroExtend(imm12:Zeros(12), datasize);
    when '1x' ReservedValue();
```

d	=	5
n	=	3
datasize	=	32
imm	=	0x0002a000

```
bits(datasize) result;
bits(datasize) operand1 = if n == 31 then SP[] else X[n];

(result, -) = AddWithCarry(operand1, imm, '0');

if d == 31 then
    SP[] = result;
else
    X[d] = result;
```



sf = ‘0’
imm12 = 0x02a
shift = ‘01’
Rd = ‘00101’
Rn = ’00011’

```
integer d = UInt(Rd);
integer n = UInt(Rn);
integer datasize = if sf == '1' then 64 else 32;
bits(datasize) imm;

case shift of
    when '00' imm = ZeroExtend(imm12, datasize);
    when '01' imm = ZeroExtend(imm12:Zeros(12), datasize);
    when '1x' ReservedValue();
```

d = 5
n = 3
datasize = 32
imm = 0x0002a000

```
bits(datasize) result;
bits(datasize) operand1 = if n == 31 then SP[] else X[n];

(result, -) = AddWithCarry(operand1, imm, '0');

if d == 31 then
    SP[] = result;
else
    X[d] = result;
```

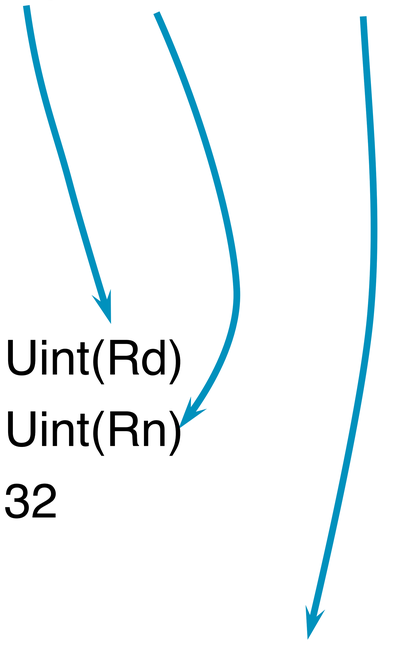
operand1 = 0x00000045
result = 0x0002a045
X[5] = 0x0002a045

sf	=	‘0’			
imm12	=	0x02a	Rd	Rn	imm12
shift	=	‘01’			
Rd	=	‘00101’			
Rn	=	’00011’			

```
integer d = UInt(Rd);
integer n = UInt(Rn);
integer datasize = if sf == '1' then 64 else 32;
bits(datasize) imm;

case shift of
    when '00' imm = ZeroExtend(imm12, datasize);
    when '01' imm = ZeroExtend(imm12:Zeros(12), datasize);
    when '1x' ReservedValue();
bits(datasize) result;
bits(datasize) operand1 = if n == 31 then SP[] else X[n];

(result, -) = AddWithCarry(operand1, imm, '0');

if d == 31 then
    SP[] = result;
else
    X[d] = result;
```

```
integer d = UInt(Rd);
integer n = UInt(Rn);
integer datasize = if sf == '1' then 64 else 32;
bits(datasize) imm;

case shift of
    when '00' imm = ZeroExtend(imm12, datasize);
    when '01' imm = ZeroExtend(imm12:Zeros(12), datasize);
    when '1x' ReservedValue();
bits(datasize) result;
bits(datasize) operand1 = if n == 31 then SP[] else X[n];

(result, -) = AddWithCarry(operand1, imm, '0');

if d == 31 then
    SP[] = result;
else
    X[d] = result;
```

sf	=	‘0’		
imm12	=	0x02a	Rd Rn	imm12
shift	=	‘01’		
Rd	=	‘00101’		
Rn	=	’00011’		
d	=	5	Uint(Rd)	
n	=	3	Uint(Rn)	
datasize	=	32	32	
imm	=	0x0002a000	ZeroExtend(imm12, 32)	



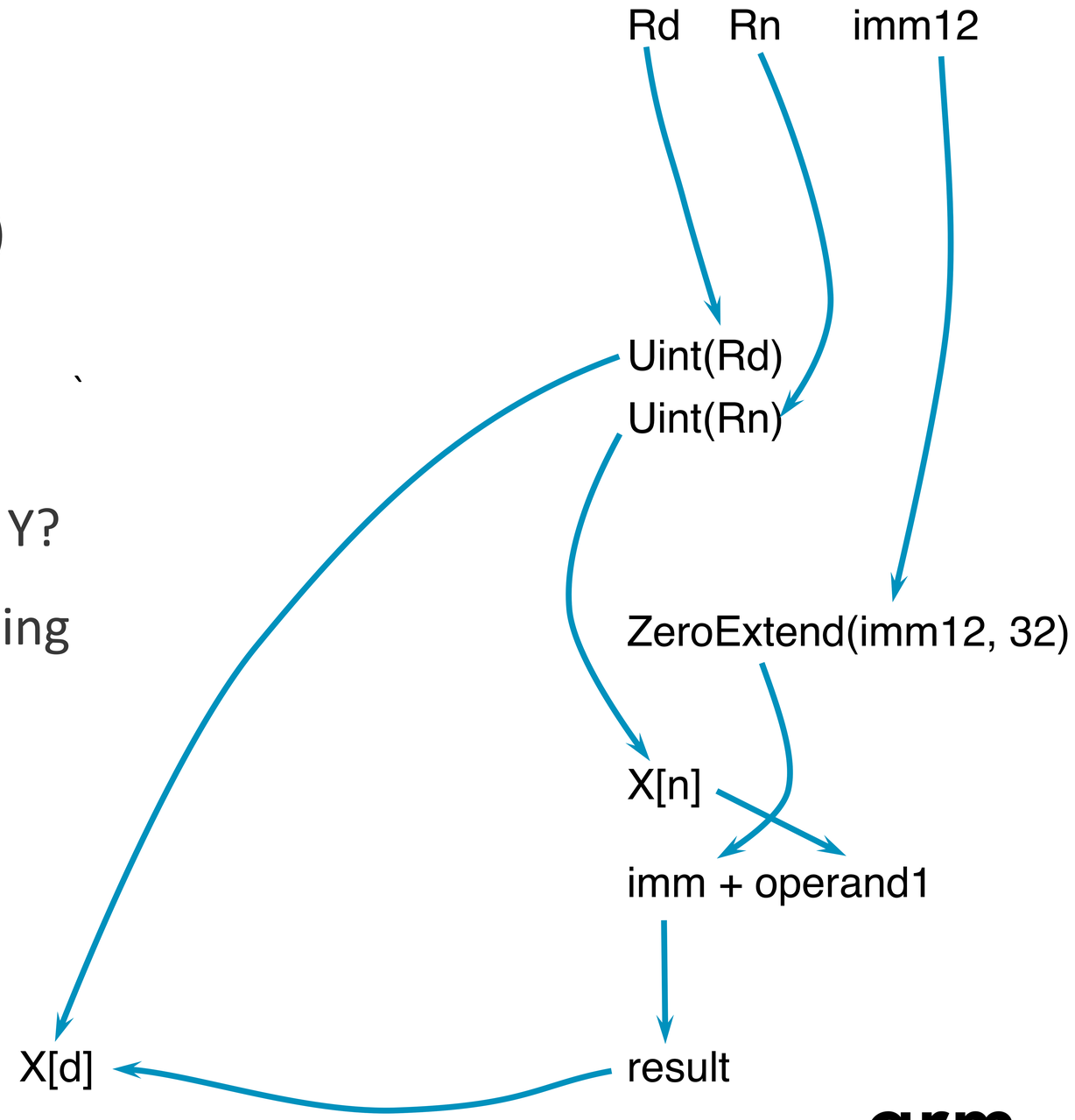
```
integer d = UInt(Rd);
integer n = UInt(Rn);
integer datasize = if sf == '1' then 64 else 32;
bits(datasize) imm;

case shift of
    when '00' imm = ZeroExtend(imm12, datasize);
    when '01' imm = ZeroExtend(imm12:Zeros(12), datasize);
    when '1x' ReservedValue();
bits(datasize) result;
bits(datasize) operand1 = if n == 31 then SP[] else X[n];

(result, -) = AddWithCarry(operand1, imm, '0');

if d == 31 then
    SP[] = result;
else
    X[d] = result;
```

sf	=	'0'	
imm12	=	0x02a	Rd Rn imm12
shift	=	'01'	
Rd	=	'00101'	
Rn	=	'00011'	
d	=	5	Uint(Rd)
n	=	3	Uint(Rn)
datasize	=	32	32
imm	=	0x0002a000	ZeroExtend(imm12, 32)
operand1	=	0x00000045	X[n]
result	=	0x0002a045	imm + operand1
X[d]	=	0x0002a045	result

Symbolic Representation

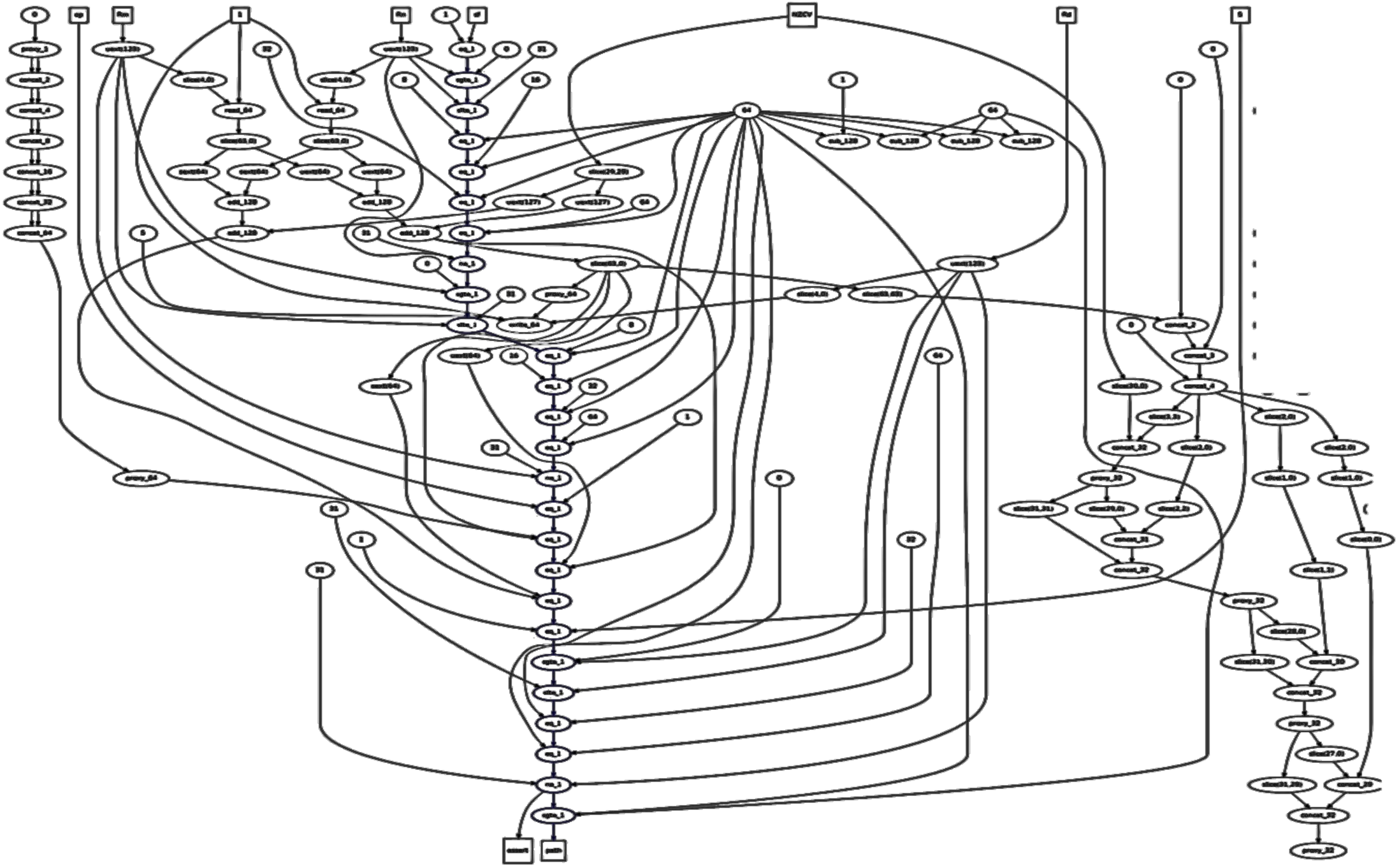
Feed to constraint solver (e.g., Z3 SMT Solver)

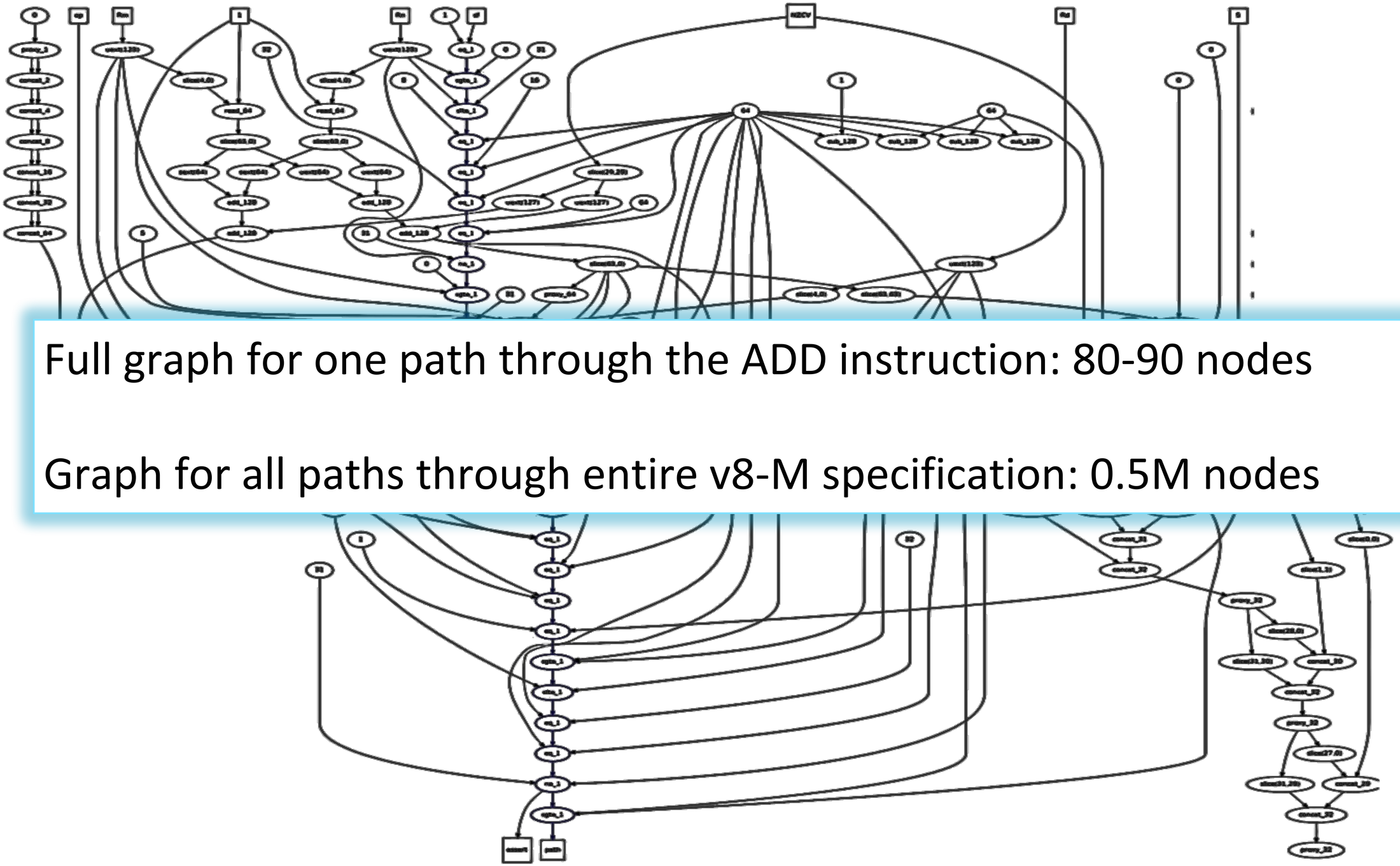
- What is the output given input Y?
- What input X produces output Y?
- What input X produces intermediate value Y?
- Generate a test input that shows X happening
  - Cf. KLEE LLVM symbolic execution



https://alastairreid.github.io/validating-specs/

arm

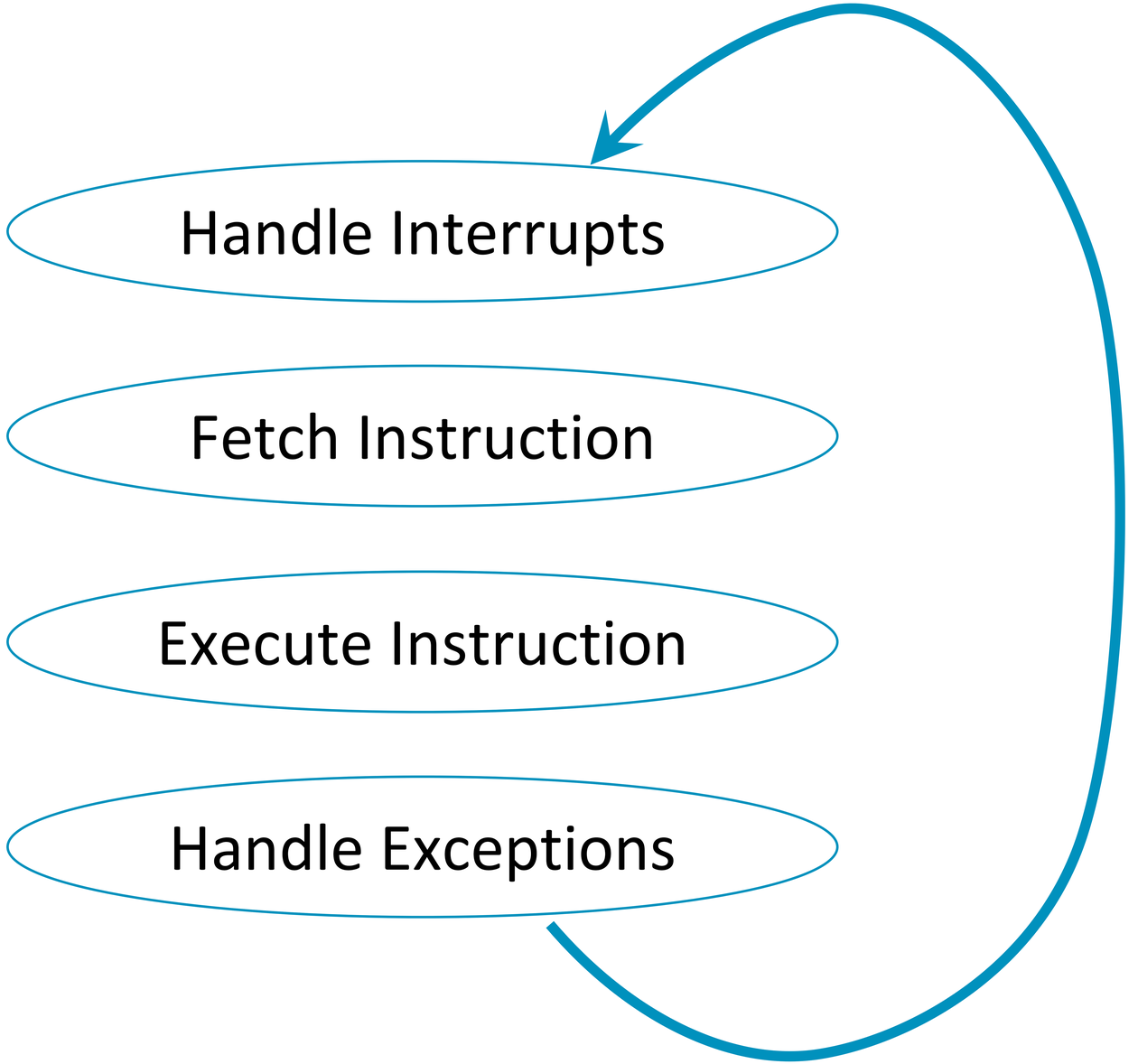




Full graph for one path through the ADD instruction: 80-90 nodes

Graph for all paths through entire v8-M specification: 0.5M nodes

From instructions to programs…



arm

Architectural Conformance Suite

https://alastairreid.github.io/papers/FMCAD_16/

Processor architectural compliance sign-off

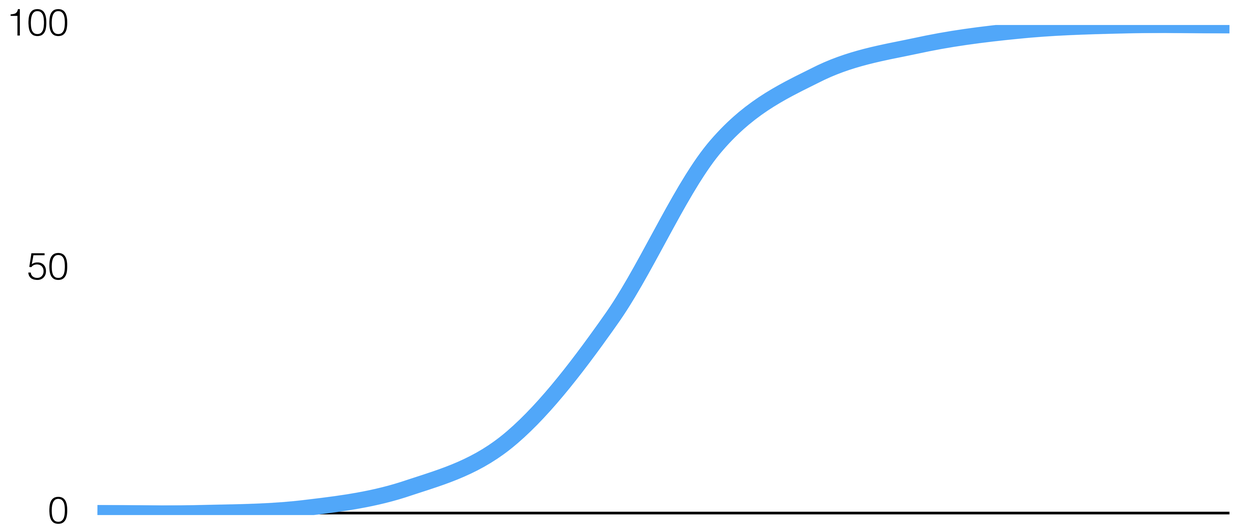
Large

- 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
- …

arm

Fuzz testing Arm binaries

External fuzzing

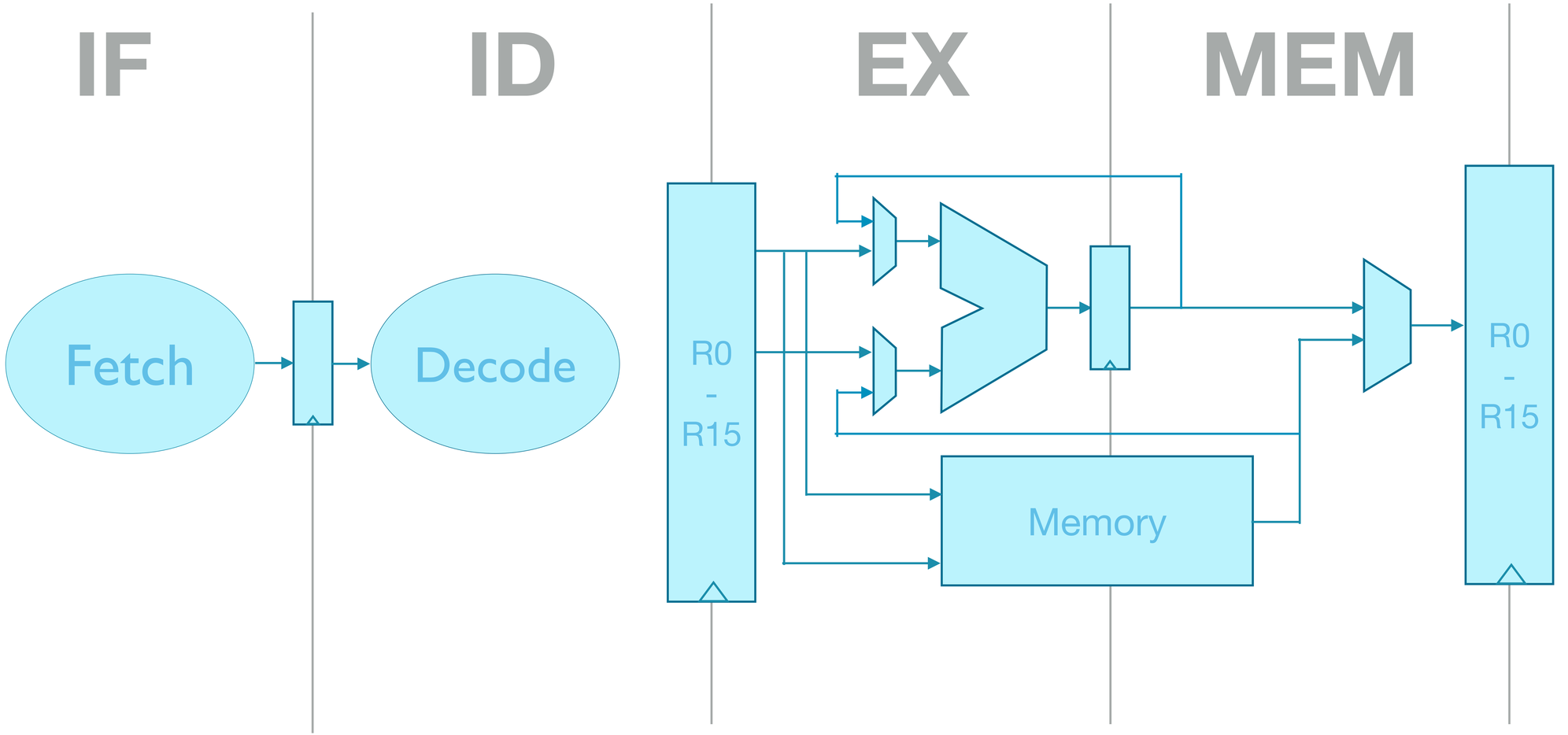
- Branches in Arm binary used to guide fuzz tester's choice of inputs
- Finds explicit control flow

Internal fuzzing

- Branches in Arm specification used to guide fuzz tester's choice of inputs
- Finds implicit control flow

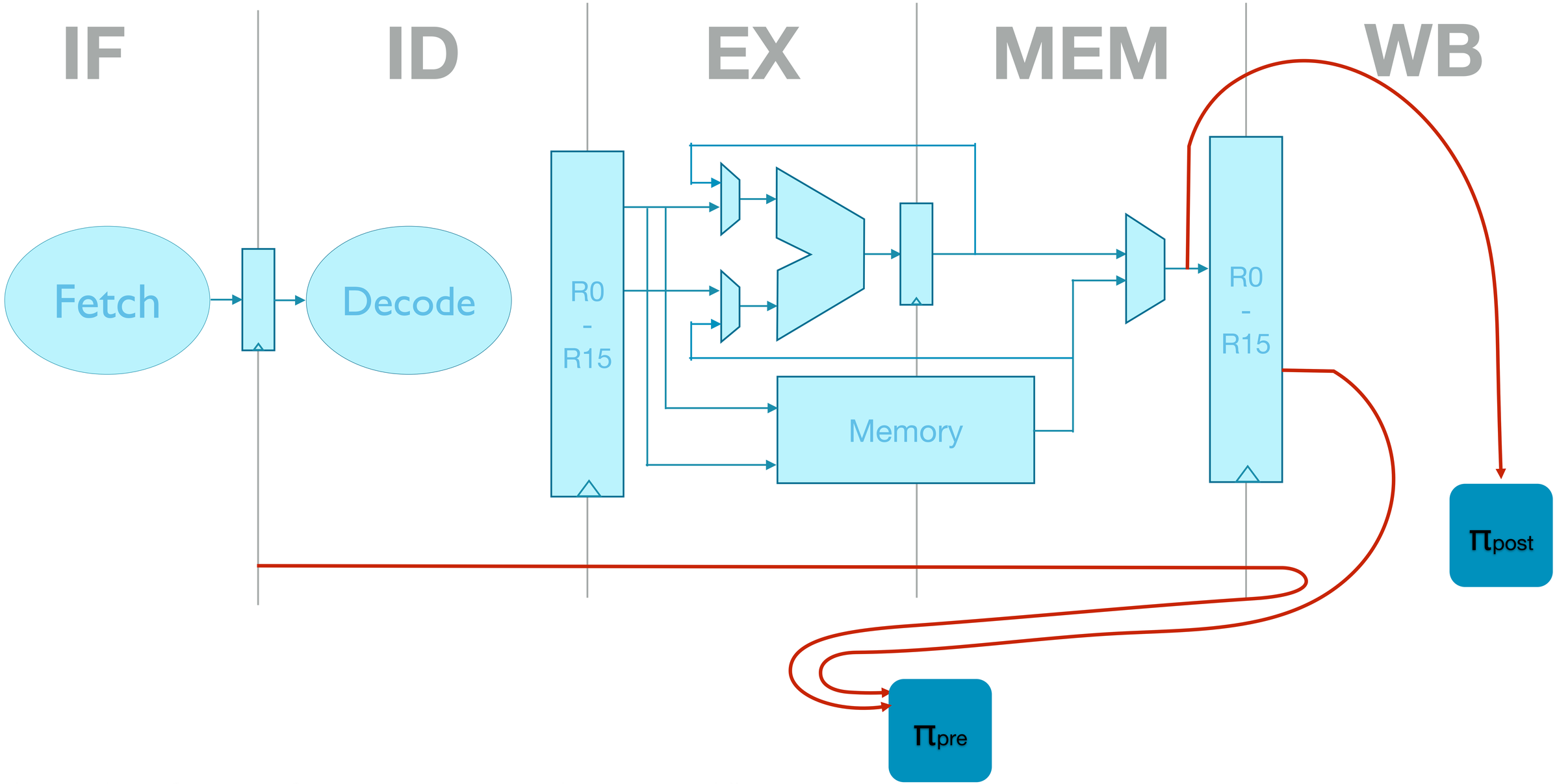
(Symbolic execution to escape plateaus)

arm



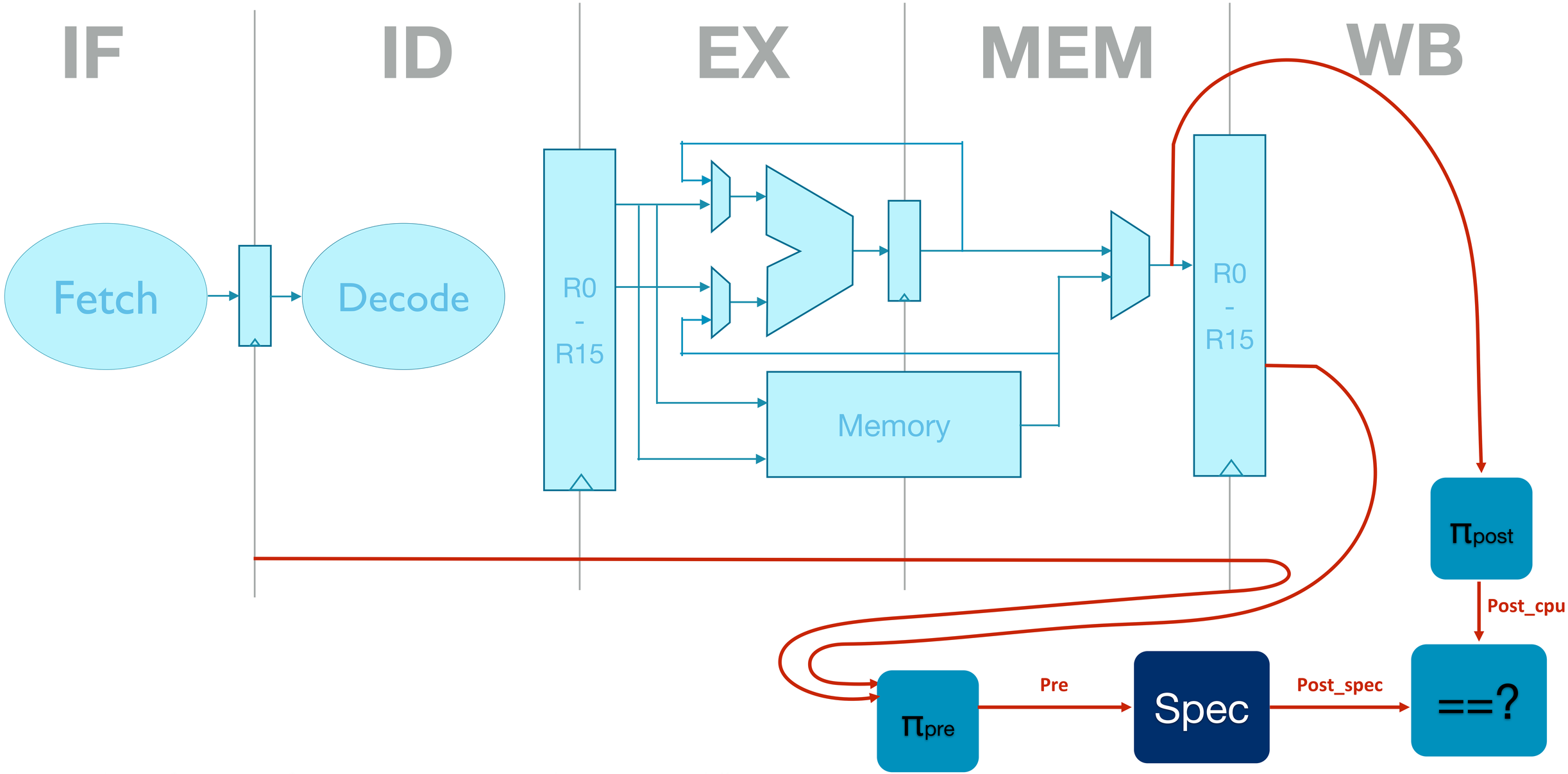
“End to End Verification of ARM processors with ISA Formal,” CAV 2016

cf “End-to-end formal ISA verification of RISC-V processors with riscv-formal”, Saal Clarke, 1pm 27th December



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

cf “End-to-end formal ISA verification of RISC-V processors with riscv-formal”, Saal Clarke, 1pm 27th December



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

cf "End-to-end formal ISA verification of RISC-V processors with riscv-formal", Saal Clarke, 1pm 27th December

Do something awesome!

Known to work

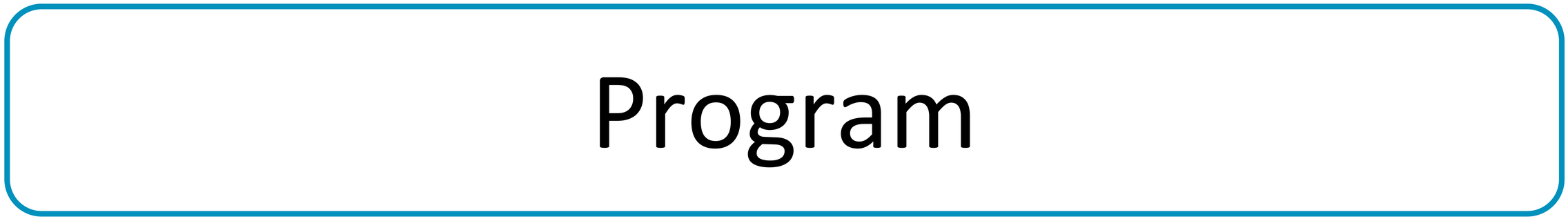
- Assembler/disassembler
- Interpreter
- Symbolic evaluation
- Generate testcases
- Fuzzing with internal feedback
- Formally validate processor design

“Should” work

- System register plugin
- Fuzzing with symbolic execution
- (Information flow analysis)
- (Test LLVM IR → ARM backend)
- (Superoptimizer http://www.eecs.qmul.ac.uk/~gretay/papers/onward2017.pdf)
- (Convert to Coq/HOL/ACL2)

arm

How can you trust formally verified software?



More formal despair: Denning, Fonseca et al.
More formal hope: Hyperkernel, Yggdrasil, Milawa, Fiat

arm

How can you trust formally verified software?

Program Specification

Program

More formal despair: Denning, Fonseca et al.
More formal hope: Hyperkernel, Yggdrasil, Milawa, Fiat

arm

How can you trust formally verified software?

Program Specification

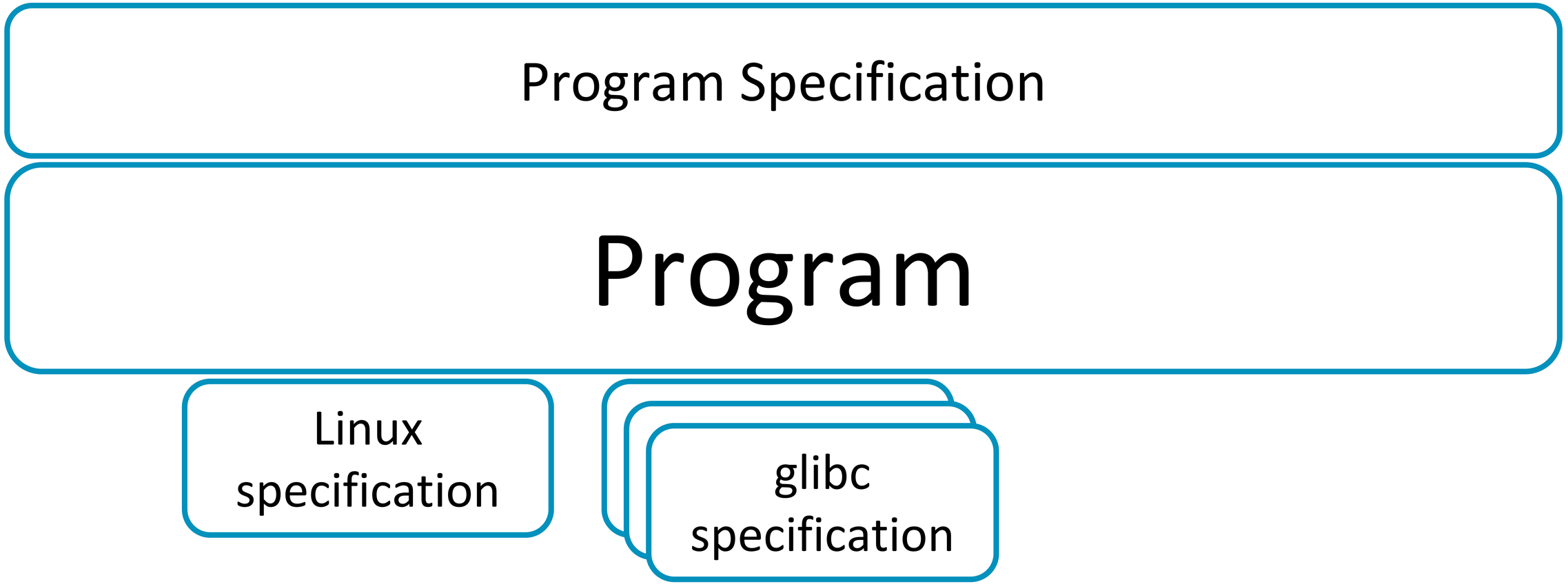
Program

Linux specification

More formal despair: Denning, Fonseca et al.
More formal hope: Hyperkernel, Yggdrasil, Milawa, Fiat

arm

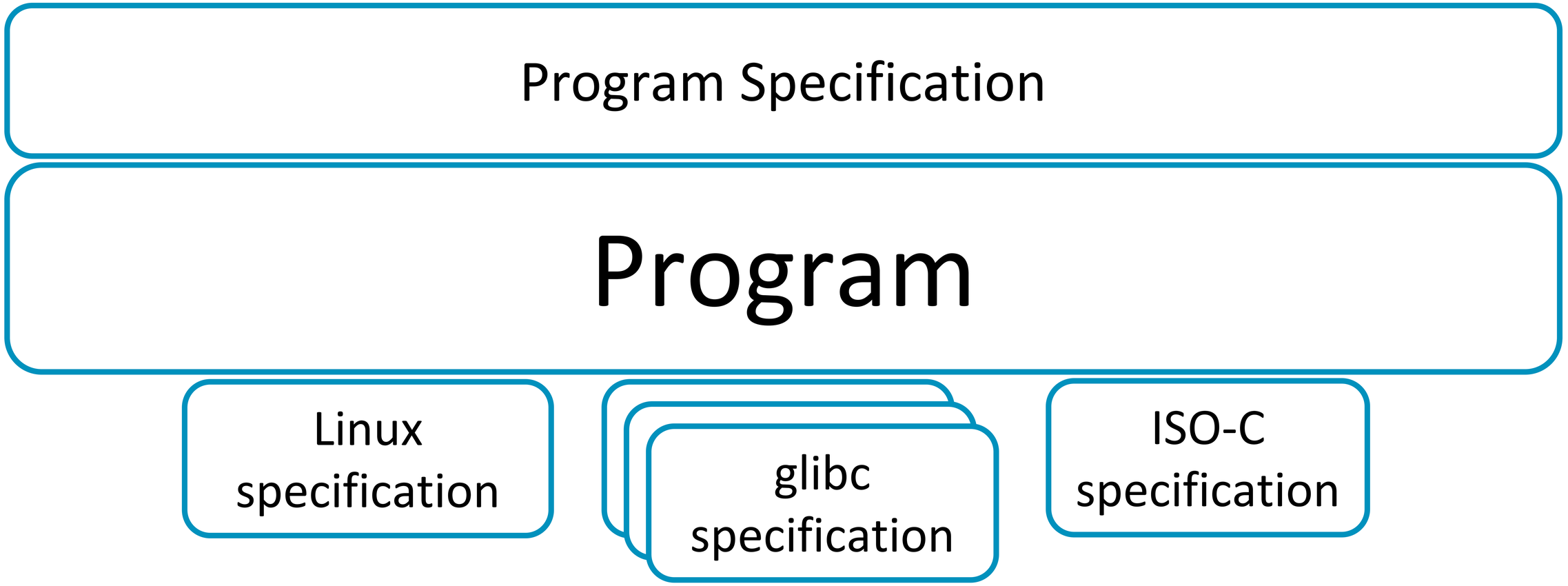
How can you trust formally verified software?



More formal despair: Denning, Fonseca et al.
More formal hope: Hyperkernel, Yggdrasil, Milawa, Fiat

arm

How can you trust formally verified software?



More formal despair: Denning, Fonseca et al.
More formal hope: Hyperkernel, Yggdrasil, Milawa, Fiat

arm

Do something awesome with the spec

Ask me questions alastair.reid@arm.com @alastair_d_reid https://alastairreid.github.io

Talk to me or Milosch Meriac (@FoolsDelight) about white hacker jobs at ARM

Thanks to those who helped get here

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)

arm