



© 2017 Arm Limited



34th Chaos Communication Congress













How Can You Trust Formally Verified Software?



Alastair Reid @alastair_d_reid ARM Research

Arm Processor Architecture

Important to understand what they do

© 2017 Arm Limited

- Widely used in many different areas: phones, tablets, IoT, HDD, ...
- Important to be able to analyse malware, security analysis, etc.

April 2017: Public release in machine readable form

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

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

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



What can you do with an executable processor specification How can you trust formally verified software?

© 2017 Arm Limited

drm

ARM Machine Readable Architecture Specification

Instructions Security features: memory protection, exceptions, privilege checks, TrustZone, ...

Links

- -
- -
- Papers
 - -
 - —
 - -

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

"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



Orm



ITD, bit [7] IT Disable. Disables some uses of IT instructions at PL1 and PL0.

In ARMV8.3 and ARMV8.2:
Load Multiple and Store Multiple Atomicity and Ordering linable. When the ornovat, feature ARMV8.2-LSMAOC is implemented, defined values are:

 In ARMV8.2:

 Load Multiple and Store Multiple Atomicity and Ordering linable. When the ornovat, feature ARMV8.2-LSMAOC is implemented, defined values are:
 Implemented atomicity and Ordering linable. When the ornovat, feature ARMV8.2-LSMAOC is implemented, defined values are:
 Implemented atomicity and Ordering linable. The ornor accesses are not required to be ordered.

 0
 accesses, and the memory accesses are not required to be ordered.

 1
 The ordering and interrupt behavior of A32 and T32 Load Multiple and Store Multiple at EL1 or EL0 is as defined for ARMv8.0.

 This bit is permitted to be cached in a TLB.
 If this bit is not implemented, it is seal.

 When this register has an architecturally-defined reset value, this field resets to 1.

Cacheability control, for data accesses at EL1 and EL0:

 C
 Meaning

 0
 All data access to Normal memory from PL1 and PL0, and all accesses to the PL1&60 stage 1 translation tables, are Non-cacheable for all levels of data and unified cache.

 1
 All data access to Normal memory from PL1 and PL0, and all accesses to the PL1&60 stage 1 translation tables, can be cached at all levels of data and unified cache.

 The PE ignores SCLTR.C for Non-secure state and data accesses to Normal memory from EL1 and EL0 are Cacheable if either:

 Alignment check enable. This is the enable bit for Alignment fault checking at PL1 and PL0:

 A

 Meaning

 0
 Alignment fault checking disabled when executing at PL1 or PL0.

 1
 Alignment fault checking disabled when executing at PL1 or PL0.

 1
 Alignment fault checking and the second registers, which has load acquire/store release, do not check that the address being executed in the land or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed (first checking enables), which is taken as a Data Abort exception.

 1
 Load/store exclusive and load-acquire/store-release instructions have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed (first check regardless of the value of the A bit.

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

In the Non-secure state the PE behaves as if the value of the SCTLR.M field is 0 for all purposes other than returning the value of a direct read of the field if either:

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

 Accessing the SCTLR

 This register can be read using MRC with the following syntax: MRC <ayntax>

 MCR <ayntax>

 This register can be written using MCR with the following syntax: MCR <ayntax>

 MCR <ayntax>

 This syntax uses the following encoding in the System instruction encoding space:

 Cara
 coproc
 CRm

 p15.0.
 Control
 Ocener CRm

 Distribution of the System instruction encoding space:

 The register is accessible as the CP1SDISABLE signal is asserted HIGH.

 Onfiguration
 Output
 Output

For a description of the prioritization of any generated exceptions, see section G1.11.2 (Exception priority order) in the ARM[®] Architecture Reference Manual, ARMv8, for ARMv8. A architecture profile for exceptions take AArch32 state, and section D1.13.2 (Synchronous exception prioritization) for exceptions taken to AArch64 state. Subject to the prioritization rules, the following traps and enables are applicable when accessing this regis

In ARMV8.1 and ARMV8.0:
Reserved, ass1.
TLSMD, bit [3]
In ARMV8.3 and ARMV8.2:
In ARMV8.3 and ARMV8.3:
In this transmost accesses by 3.32 and T32 Load Multiple and Store Multiple at EL1 or EL0 that are marked at stage 1 as Device-nGRE/DeviceIn ARMV9.3:
In this bit is not implemented, it is stall.
If this bit is not implemented, it is stall.
When this register has an architecturally-defined reset value, this field resets to 1.

In ARMv8.1 and ARMv8.0:

In ARMv8.1 and ARMv8.0:

 EL2 is using AArch32 and the value of <u>HCR.DC</u> is 1.
 EL2 is using AArch64 and the value of <u>HCR_EL2.DC</u> is 1. When this register has an architecturally-defined reset value, this field resets to 0.

M, bit [0]

Alignment check enable. This is the enable bit for Alignment fault checking at PL1 and PL0:

MUU enable for III.1 and III.0 stage 1 address translation. Possible values of this bit are:

 M
 Meaning

 0
 IEI.4 and IEI.0 stage 1 address translation disabled.

 See the SCTLR.1 field for the bhavior of instruction accesses to Normal memory.

 1
 IE.1 and IEI.0 stage 1 address translation enabled.

 EL2 is using AArch32 and the value of <u>HCR.</u>{DC, TGE} is not {0, 0}.
 EL2 is using AArch64 and the value of <u>HCR_EL2.</u>{DC, TGE} is not {0, 0}. When this register has an architecturally-defined reset value, this field resets to 0.

Traps and enables

For a description of the prioritization of any generated exceptions, see section GLTL2 (Exceptions taken to AArch64 state, Subject to When EL2 is implemented and is using AArch64 and SCR_EL3NS=1 && HCR_EL2E2H=0:
If HCR_EL2_TVM=1, Non-secure write accesses to this register from EL1 are trapped to EL2.
If HCR_EL2_TVM=1, Non-secure accesses to this register from EL1 are trapped to EL2.
If HCR_EL2_TVM=1, Non-secure write accesses to this register from EL1 are trapped to EL2.
If HCR_EL2_TVM=1, Non-secure accesses to this register from EL1 are trapped to EL2.
If HCR_EL2_TVM=1, Non-secure accesses to this register from EL1 are trapped to EL2.
If HCR_EL2_TVM=1, Non-secure write accesses to this register from EL1 are trapped to EL2.
If HCR_EL2_TVM=1, Non-secure write accesses to this register from EL1 are trapped to EL2.
If HCR_EL2_TVM=1, Non-secure write accesses to this register from EL1 are trapped to EL2.
If HCR_EL2_TVM=1, Non-secure write accesses to this register from EL1 are trapped to EL2.
If HCR_EL2_TVM=1, Non-secure write accesses to this register from EL1 are trapped to EL2.
If HCR_EL2_TVM=1, Non-secure read accesses to this register from EL1 are trapped to EL2.
If HCR_EL2_TVM=1, Non-secure read accesses to this register from EL1 are trapped to EL2.
If HCR_EL2_TVM=1, Non-secure write accesses to this register from EL1 are trapped to EL2.
If HCR_EL2_TVM=1, Non-secure write accesses to this register from EL1 are trapped to HL2.

Reserved, and I. C, bit [2] Cacheability control, for data accesses at HL1 and HL0:

A, bit [1]

 Tot be construction of the construction of



The SCTLR characteristics are:

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

SCTLR, System Control Register

Orm





The SCTLR characteristics are:

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

Field descriptions

The SCTLR bit assignments are:

			1000					20		
TRE	0	0	EE	0	SPAN	1	0	UWXN	WXN	n٦

SCTLR, System Control Register

17 16 15 14 13 12 11 10 9 8 7 6 18 TWE 0 nTWI 0 0 V I 1 0 0 SED ITD UNK CP15BEN LSMAOE nTLSMD C A M









The SCTLR characteristics are:

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

Field descriptions

The SCTLR bit assignments are:

28 27 26 25 24 23 22 21 20 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:

XN	Μ
)	This control has no effect on m
L	Any region that is writable in the
	forced to XN for accesses from

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.

SCTLR, System Control Register

18 15 14 13 12 11 10 9 16 0 TEAFETRE 0 0 EE 0 SPAN 1 0 UWXNWXNNTWE 0 NTW 0 0 V I 1 0 0 SED TD UNK CP15BEN LSMAOE NTLSMD C A N

leaning

nemory access permissions. the PL1&0 translation regime is n software executing at PL1 or PL0.





© 2017 Arm Limited

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



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

© 2017 Arm Limited

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



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

SCTLR.WXN = 1;

© 2017 Arm Limited

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



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

SCTLR.WXN = 1;

© 2017 Arm Limited

Write permission im	plies XN
all memory regions	that are v

WXN

This control has r
permissions.
Any region that is
regime is forced t
executing at PL1

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

N (Execute-never). For the PL1&0 translation regime, this bit can force writable to be treated as XN. The possible values of this bit are:

Meaning

no effect on memory access

is writable in the PL1&0 translation to XN for accesses from software or PLO.



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).

_	30									20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
8f	0	Ô	1	0	Ô	Û	1	8	ift					imr	n12								Rn					Rd		
	ор	S																												

32-bit (sf == 0)

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

64-bit (sf == 1)

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

```
integer d = <u>UInt(Rd);</u>
 integer n = UInt(Rn);
integer datasize = if sf == '1' then 64 else 32;
bits(datasize) imm;
case shift of
    when '00' imm = <u>ZeroExtend(imm12, datasize);</u>
    when '01' imm = <u>ZeroExtend(imm12:Zeros(12)</u>, datasize);
   when 'lx' ReservedValue();
```

Assembler Symbols

	-110
<shift></shift>	Is the optional left shift to apply to the immediate, defaulting to LSL #0 and encoded in "shift":
<imm></imm>	Is an unsigned immediate, in the range 0 to 4095, encoded in the "imm12" field.
<xnisp></xnisp>	Is the 64-bit name of the source general-purpose register or stack pointer, encoded in the "Rn" field.
<xdisp></xdisp>	Is the 64-bit name of the destination general-purpose register or stack pointer, encoded in the "Rd" field.
<walwsp></walwsp>	Is the 32-bit name of the source general-purpose register or stack pointer, encoded in the "Rn" field.
<wdiwsp></wdiwsp>	Is the 32-bit name of the destination general-purpose register or stack pointer, encoded in the "Rd" field.

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

Alias Conditions

Alias Is preferred when

MOV (to/from SP) shift == '00' &&	imm12 == '00000000000'	&& (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;
```

ADD (immediate)



32-bit (sf === 0)

64-bit (sf === 1)

<WdWSP>





destination register.

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

29	28	27	26	25	24	23	22	21	20	19
0	1	0	0	0	1	sh	ift			
S										



Assembler Symbols

in the "Rd" field.

the "Rn" field.

in the implied for the

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

15 16 imm12

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

ADD $\langle Xd | SP \rangle$, $\langle Xn | SP \rangle$, $\# \langle imm \rangle \{$, $\langle shift \rangle \}$

Is the 32-bit name of the destination general-purpose register or stack pointer, encoded

Is the 32-bit name of the source general-purpose register or stack pointer, encoded in

Is the 64-bit name of the destination general-purpose register or stack pointer, encoded



Assembler / Disassembler

op S

<->

where

Xd|SP = RegXSP(UInt(Rd));

<Xn | SP > = RegXSP(UInt(Rn));

= UInt(imm12);<imm>

<shift> = Optional("LSL #0",

case shift {

© 2017 Arm Limited





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

] <imm> " "</imm>		11	11	V V	VV	<sł< th=""></sł<>
-------------------	--	----	----	------------	----	-------------------

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



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).

29 28 27 26 25 24 31 30 0 0 0 1 0 sf op S

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

case shif	t of				
when	'00'	imm	=	Zε	er
when	'01'	imm	=	Zε	er
when	'1x'	Rese	erv	<u>rec</u>	lV
bits(datas	ize)	resu	lt	;	
bits(datas	ize)	oper	an	d1	=

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

if d == 31 then

SP[] = result;

else

X[d] = result;

© 2017 Arm Limited

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7
sh	nift						imn	n12								Rn

roExtend(imm12, datasize); roExtend(imm12:Zeros(12), datasize); Value();

= if n == 31 then SP[] else X[n];

https://www.meriac.com/archex/A64 v83A ISA/add addsub imm.xml





31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 sf 0 0 1 0 0 0 1 shift im op S

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

case shift of	
when '00'	imm = <u>ZeroExte</u>
when '01'	imm = <u>ZeroExte</u>
when '1x'	<u>ReservedValue(</u>
<pre>bits(datasize)</pre>	result;
<pre>bits(datasize)</pre>	operand1 = if r

(result, -) = <u>AddWithCarry(operand1, imm, '0');</u>

if d == 31 then SP[] = result;

else

X[d] = result;

6151413121110	9	8	7	6	5	4	3	2	1	0
1 m12		F	Rn					Rd		

end(imm12, datasize); end(imm12:Zeros(12), datasize););

n == 31 then <u>SP[]</u> else <u>X[n]</u>;

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 sf 0 0 1 0 0 0 1 shift im op S

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

case shift of	
when '00'	imm = <u>ZeroExte</u>
when '01'	imm = <u>ZeroExte</u>
when '1x'	<u>ReservedValue(</u>
<pre>bits(datasize)</pre>	result;
<pre>bits(datasize)</pre>	operand1 = if r

(result, -) = <u>AddWithCarry(operand1, imm, '0');</u>

if d == 31 then SP[] = result;

else

X[d] = result;

6151413121110	9	8	7	6	5	4	3	2	1	0
1 m12		F	Rn					Rd		

end(imm12, datasize); end(imm12:Zeros(12), datasize););

n == 31 then <u>SP[]</u> else <u>X[n]</u>;

'0'
0x02a
'01'
'00101'
'00011'

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 sf 0 0 1 0 0 0 1 shift im op S

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

case shift of	
when '00'	imm = <u>ZeroExte</u>
when '01'	imm = <u>ZeroExte</u>
when '1x'	<u>ReservedValue(</u>
<pre>bits(datasize)</pre>	result;
<pre>bits(datasize)</pre>	operand1 = if r

(result, -) = <u>AddWithCarry(operand1, imm, '0');</u>

if d == 31 then SP[] = result;

else

X[d] = result;

6151413121110	9	8 7	6	5	4	3	2	1	0
ւm12		R	n				Rd		

end(imm12, datasize); end(imm12:<u>Zeros(12), datasize);</u>);

n == 31 then <u>SP[]</u> else <u>X[n]</u>;

Sf	' 0'
imm12	0x02a
shift	'01'
Rd	'00101'
Rn	'00011'
d	5
n	3
datasize	32
imm	0x0002a000

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 sf 0 0 1 0 0 0 1 shift îm op S

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

case shift of	
when '00'	imm = <u>ZeroExte</u>
when '01'	imm = <u>ZeroExte</u>
when '1x'	<u>ReservedValue(</u>
<pre>bits(datasize)</pre>	result;
<pre>bits(datasize)</pre>	operand1 = if r

(result, -) = <u>AddWithCarry(operand1, imm, '0');</u>

if d == 31 thenSP[] = result;

else

X[d] = result;

6151413121110	9	8 7	6	5	4	3	2	1	0
ւm12		R	n				Rd		

end(imm12, datasize); = 0x0002a000end(imm12:Zeros(12), datasize); IMM);

n == 31 then SP[] else X[n];

Sf	' 0'
imm12	0x02a
shift	'01'
Rd	'00101'
Rn	'00011'
d	5
d	5 3
d n datasize	5 3 32

operand1 = 0x0000045

= 0x0002a045result

X[5] = 0x0002a045

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 = <u>ZeroExtend(imm12, datasize);</u> when '01' imm = <u>ZeroExtend(imm12:Zeros(12)</u>, datasize); when 'lx' ReservedValue(); bits(datasize) result; bits(datasize) operand1 = if n == 31 then SP[] else X[n];

(result, -) = <u>AddWithCarry(operand1, imm, '0');</u>

if d == 31 then $\underline{SP}[] = result;$ else

X[d] = result;

'0'
0x02a
'01'
'00101'
'00011'

Rn imm12Rd

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

case shift of when '00' imm = <u>ZeroExtend(imm12, datasize);</u> 0x0002a000 when '01' imm = <u>ZeroExtend(imm12:Zeros(12)</u>, datasize); IMM when 'lx' <u>ReservedValue();</u> bits(datasize) result; bits(datasize) operand1 = if n == 31 then SP[] else X[n];

(result, -) = <u>AddWithCarry(operand1, imm, '0');</u>

if d == 31 then $\underline{SP}[] = result;$ else

X[d] = result;

Sf	' 0'
imm12	0x02a
shift	'01'
Rd	'00101'
Rn	'00011'
d	5
n	3
datasize	32



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

case shift of when '00' imm = <u>ZeroExtend(imm12, datasize);</u> when '01' imm = <u>ZeroExtend(imm12:Zeros(12)</u>, datasize); IMM when 'lx' <u>ReservedValue();</u> 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;





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/



Uint(Rn)

Rn

Rd

ZeroExtend(imm12, 32)

imm12

imm + operand1

result







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...





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



https://alastairreid.github.io/papers/FMCAD 16/

Orm



100

50



© 2017 Arm Limited

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



Fuzz testing Arm binaries

External fuzzing

- Finds explicit control flow

Internal fuzzing

- Finds implicit control flow

(Symbolic execution to escape plateaus)

• Branches in Arm binary used to guide fuzz tester's choice of inputs

• Branches in Arm specification used to guide fuzz tester's choice of inputs







"End to End Verification of ARM processors with ISA Formal," <u>CAV 2016</u> cf "End-to-end formal ISA verification of RISC-V processors with riscv-formal", Saal Clarke, 1pm 27th December





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



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 \rightarrow ARM backend)
- (Superoptimizer

http://www.eecs.qmul.ac.uk/~gretay/papers/onward2017.pdf

- (Convert to Coq/HOL/ACL2)







Program

More formal despair: <u>Denning</u>, <u>Fonseca et al.</u> More formal hope: <u>Hyperkernel</u>, <u>Yggdrasil</u>, <u>Milawa</u>, <u>Fiat</u>











How can you trust formally verified software?

Program Specification

Program

More formal despair: <u>Denning</u>, <u>Fonseca et al.</u> More formal hope: <u>Hyperkernel</u>, <u>Yggdrasil</u>, <u>Milawa</u>, <u>Fiat</u>











How can you trust formally verified software?

Program Specification

Program

Linux specification

> More formal despair: <u>Denning</u>, <u>Fonseca et al.</u> More formal hope: <u>Hyperkernel</u>, <u>Yggdrasil</u>, <u>Milawa</u>, <u>Fiat</u>











How can you trust formally verified software?

Program Specification

Program

Linux specification

More formal despair: <u>Denning</u>, <u>Fonseca et al.</u> More formal hope: <u>Hyperkernel</u>, <u>Yggdrasil</u>, <u>Milawa</u>, <u>Fiat</u>

specification











How can you trust formally verified software?

Program Specification

Program

Linux specification

More formal despair: <u>Denning</u>, Fonseca et al. More formal hope: <u>Hyperkernel</u>, <u>Yggdrasil</u>, <u>Milawa</u>, <u>Fiat</u>

BIINC specification







Do something awesome with the spec Ask me questions <u>alastair.reid@arm.com</u> <u>@alastair d reid</u> <u>https://alastairreid.github.io</u> Talk to me or Milosch Meriac (<u>@FoolsDelight</u>) about <u>white hacker jobs</u> at ARM Thanks to those who helped get here

Alasdair Armstrong (Cambridge U.) Curtis Dunham (ARM) Alex Chadwick (ARM) David Gilday (ARM) David Hoyes (ARM) Ali Zaidi (ARM) Anastasios Deligiannis (ARM) David Seal (ARM) Anthony Fox (Cambridge U.) Daniel Bailey (ARM) Ashan Pathirane (ARM) Erin Shepherd (ARM) Francois Botman (ARM) Belaji Venu (ARM) Bradley Smith (ARM) George Hawes (ARM) Graeme Barnes (ARM) Brian Foley (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)

