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/...
What can you do with an executable processor specification

How can you trust formally verified software?
ARM Machine Readable Architecture Specification

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

Links
- 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
SCTRL, System Control Register

The SCTRL characteristics are:

**Purpose**

Provides the top level control of the system, including its memory system.
SCTRL, System Control Register

The SCTRL characteristics are:

**Purpose**

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

**Field descriptions**

The SCTRL 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  TEAFETRE 0  0  EE  0  SPAN 1  0  UWXN  WXN  tTWE 0  nTWI 0  0  V  1  0  0  SED  ITD  UNK  CP15  BEN  LSM  AE  nTLSMD  C  A  M
```
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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

**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:

<table>
<thead>
<tr>
<th>WXN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>This control has no effect on memory access permissions.</td>
</tr>
<tr>
<td>1</td>
<td>Any region that is writable in the PL1&amp;0 translation regime is forced to XN for accesses from software executing at PL1 or PL0.</td>
</tr>
</tbody>
</table>

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.
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
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:

<table>
<thead>
<tr>
<th>WXN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>This control has no effect on memory access permissions.</td>
</tr>
<tr>
<td>1</td>
<td>Any region that is writable in the PL1&amp;0 translation regime is forced to XN for accesses from software executing at PL1 or PL0.</td>
</tr>
</tbody>
</table>

See also: [https://github.com/gdelugre/ida-arm-system-highlight](https://github.com/gdelugre/ida-arm-system-highlight)
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).

| 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 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| sf | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | shift | imm12 |       | Rn |       | Rd |

32-bit (sf == 0)

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

64-bit (sf == 1)

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

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.
Assembler / Disassembler

```
[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();
            });
```

See also: [https://github.com/agustingianni/retools](https://github.com/agustingianni/retools) and [https://github.com/nspin/hs-arm](https://github.com/nspin/hs-arm)
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)**.

<table>
<thead>
<tr>
<th>sf</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>shift</th>
<th>imm12</th>
<th>Rn</th>
<th>Rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

integer d = \texttt{UINT}(Rd);
integer n = \texttt{UINT}(Rn);
integer datsize = if sf == '1' then 64 else 32;
bits(datsize) imm;

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

(result, -) = \texttt{AddWithCarry}(operand1, imm, '0');

if d == 31 then
  \texttt{SP}[] = result;
else
  \texttt{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;
```java
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;
```
X[5] = 0x0002a045

```java
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 datysize = if sf == '1' then 64 else 32;
bits(datysize) imm;

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

bits(datysize) result;
bits(datysize) 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
shift = '01'
Rd = '00101'
Rn = '00011'
d = 5
Uint(Rd)
n = 3
Uint(Rn)
datasize = 32
imm = 0x0002a000
ZeroExtend(imm12, 32)
operand1 = 0x00000045
X[n] = 0x0002a045
result = imm + operand1
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;
bites(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;
integer $d = \text{UInt}(Rd);$  
integer $n = \text{UInt}(Rn);$  
integer $\text{datasize} = \text{if sf == '1' then 64 else 32;}$  
bits($\text{datasize}$) $\text{imm};$  

\[
\text{case shift of} \\
\quad \text{when '00' imm = ZeroExtend(imm12, datasize);} \\
\quad \text{when '01' imm = ZeroExtend(imm12:Zeros(12), datasize);} \\
\quad \text{when '1x' ReservedValue();}
\]

bits($\text{datasize}$) result;  
bits($\text{datasize}$) operand1 = if $n == 31$ then $\text{SP}[]$ else $X[n];$  

\[(\text{result, -}) = \text{AddWithCarry(operand1, imm, '0');}\]

if $d == 31$ then  
\quad $\text{SP}[] = \text{result};$  
else  
\quad $X[d] = \text{result};$
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

Cf. KLEE LLVM symbolic execution

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

- Handle Interrupts
- Fetch Instruction
- Execute Instruction
- Handle Exceptions
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/
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
- ...

© 2017 Arm Limited
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)
“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)
- (Convert to Coq/HOL/ACL2)
How can you trust formally verified software?

More formal despair: Denning, Fonseca et al.
More formal hope: Hyperkernel, Yggdrasil, Milawa, Fiat
How can you trust formally verified software?

More formal despair: Denning, Fonseca et al.
More formal hope: Hyperkernel, Yggdrasil, Milawa, Fiat
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
How can you trust formally verified software?

Program Specification

Program

Linux specification

glibc specification

More formal despair: Denning, Fonseca et al.
More formal hope: Hyperkernel, Yggdrasil, Milawa, Fiat
How can you trust formally verified software?

Program Specification

Program

Linux specification

glibc specification

ISO-C specification

More formal despair: Denning, Fonseca et al.
More formal hope: Hyperkernel, Yggdrasil, Milawa, Fiat
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.)  Curtis Dunham (ARM)  Isobel Hooper (ARM)  Michele Riga (ARM)
Alex Chadwick (ARM)  David Gilday (ARM)  Jack Andrews (ARM)  Milosch Meriac (ARM)
Ali Zaidi (ARM)  David Hoyes (ARM)  Jacob Eapen (ARM)  Nigel Stephens (ARM)
Anastasios Deligiannis (ARM)  David Seal (ARM)  Jon French (Cambridge U.)  Niyas Sait (ARM)
Ashan Pathirane (ARM)  Erin Shepherd (ARM)  Krassy Gochev (ARM)  Peter Sewell (Cambridge U.)
Belaji Venu (ARM)  Francois Botman (ARM)  Lewis Russell (ARM)  Peter Vrabel (ARM)
Bradley Smith (ARM)  George Hawes (ARM)  Matthew Leach (ARM)  Richard Grisenthwaite (ARM)
Brian Foley (ARM)  Graeme Barnes (ARM)  Meenu Gupta (ARM)  Rick Chen (ARM)

© 2017 Arm Limited