T outing Large Specifications: The Virtuous Cycle

Alastair Reid
alastair.reid@arm.com
@alastair_d_reid

The Architecture for the Digital World®
History of ARM

Joint venture between Acorn Computers and Apple

1990

Designed into first mobile phones and then smartphones

1993 onwards

Now all electronic devices can use smart ARM technology

Today
ARM-based chip shipments

ARM-based chips to date

1991

2015

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15.1bn

90bn
Correctness

Portability / Predictability
Security
Commercial pressures
“It’s nice not to suck” — Adam Langley
Specifications

What to build
What to test
What to expect
Application
Library
OS
Compiler
Processor
Specifications: The New Bottleneck

Qualities of Specifications

Applicability
Scope
Trustworthiness

Testing and Using Specifications

The Virtuous Cycle
Applicability
Scope

Compiler targeted instructions?
User-level instructions?
User+Supervisor?
User+Supervisor+Hypervisor+Secure Monitor?
**ISA Specification - ASL**

**Encoding T3**

ARMv7-M

MOVE(S)<->.W <Rd>,<Rm>

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| 1  | 1  | 1  | 0  | 1  | 0  | 1  | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | | |

```

d = UInt(Rd); m = UInt(Rm); setflags = (S == '1');
if setflags && (d IN {13,15} || m IN {13,15}) then UNPREDICTABLE;
if !setFlags && (d == 15 || m == 15 || (d == 13 && m == 13)) then UNPREDICTABLE;

if ConditionPassed() then
    EncodingSpecificOperations();
    result = R[m];
    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 unchanged
        // APSR.V unchanged
```
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 Spec (lines of code)

<table>
<thead>
<tr>
<th>Category</th>
<th>v8-A</th>
<th>v8-M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instructions (Int/FP/SIMD)</td>
<td>26,000</td>
<td>6,000</td>
</tr>
<tr>
<td>Exceptions</td>
<td>4,000</td>
<td>3,000</td>
</tr>
<tr>
<td>Memory</td>
<td>3,000</td>
<td>1,000</td>
</tr>
<tr>
<td>Debug</td>
<td>3,000</td>
<td>1,000</td>
</tr>
<tr>
<td>Misc (Test support)</td>
<td>5,500</td>
<td>2,000</td>
</tr>
<tr>
<td>Total</td>
<td>43,000</td>
<td>15,000</td>
</tr>
</tbody>
</table>
# System Register Spec

<table>
<thead>
<tr>
<th></th>
<th>v8-A</th>
<th>v8-M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>586</td>
<td>186</td>
</tr>
<tr>
<td>Fields</td>
<td>3951</td>
<td>622</td>
</tr>
<tr>
<td>Constant</td>
<td>985</td>
<td>177</td>
</tr>
<tr>
<td>Reserved</td>
<td>940</td>
<td>208</td>
</tr>
<tr>
<td>Impl. Defined</td>
<td>70</td>
<td>10</td>
</tr>
<tr>
<td>Passive</td>
<td>1888</td>
<td>165</td>
</tr>
<tr>
<td>Active</td>
<td>68</td>
<td>62</td>
</tr>
<tr>
<td>Operations</td>
<td>112</td>
<td>10</td>
</tr>
</tbody>
</table>
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?
Qualities of Specifications

Testing and Using Specifications
  Testing Specifications (FMCAD 2016)
  Verifying Processors (CAV 2016)
  Generating Testcases
  Security Checking
  Booting an OS
  Fuzzing an OS

The Virtuous Cycle
Test Stimulus \rightarrow ARM Spec \rightarrow Oracle \rightarrow =?= \rightarrow Test Stimulus
Architecture 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
Testing Specifications

Test Suite → Test Program → v8-A/M Specification → ASL Interpreter/Compiler

Bug in Spec → Bug in Test
Verifying Processors

ARM Specification

ARM Processor

Model Checker

Bug in Processor

Bug in Specification

CEX

CEX

The Architecture for the Digital World®
Verification Challenges

Detect the hard-to-find bugs
Large Specifications / Processors
Fit the development flow
Memory

R0 - R15
Checking an instruction

ADD
Checking an instruction

CMP LDR  ADD  STR BNE

Context
The diagram illustrates the ARM processing pipeline, which consists of five stages:

1. **IF (Instruction Fetch)**: The instructions are fetched from memory and passed to the next stage.
2. **ID (Instruction Decode)**: The fetched instructions are decoded into executable form.
3. **EX (Execution)**: The decoded instructions are executed, and the results are sent to the Memory stage.
4. **MEM (Memory Access)**: The results are stored in memory or used as operands for further processing.
5. **WB (Write Back)**: The results are written back to the appropriate registers.

Additionally, the diagram shows the interaction between the processing stages and the Memory and Spec (speculative execution) stages, indicating how data flows through the pipeline and how speculative execution can affect the pipeline's efficiency.
## ISA-Formal Properties

<table>
<thead>
<tr>
<th></th>
<th>ADC</th>
<th>ADD</th>
<th>B</th>
<th>...</th>
<th>YIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>R[]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>NZCV</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>SP</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>PC</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>S[], D[], V[]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>FPSR</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>MemRead</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>MemWrite</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>SysRegRW</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>ELR</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>ESR</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Automation

Architecture Specification → ASL to Verilog → Combinational Verilog

- Specialize
- Monomorphize
- Constant Propagation
- Width Analysis
- Exception Handling
- ...

The Architecture for the Digital World®
Verification Progress

Bugs Found (%) vs Time (weeks)
Verifying Processors

- v8-A/M Spec (ASL)
- ASL to Verilog
- v8-A/M Spec (Verilog)
- Model Checker
- ARM Processor

- Bug in Processor
- Bug in Specification
Testcase Generation

TestCase → v8-A/M Specification → ASL Interpreter → Branch Coverage

Symbolic Dataflow Graph → SMT Solver → TestCase
Security Checking

Test Program
v8-M Specification
ASL Interpreter

Symbolic Dataflow Graph

Information Flow
Booting an OS

- Application
- mbed OS
- v8-M Specification
- ASL Interpreter/Compiler
Fuzzing the mbed OS

Test Case

Random Application

mbed OS

v8-M Specification

ASL Interpreter/Compiler

AFL Fuzzer

Crash/Fail

Branch Coverage
Creating a Virtuous Cycle

- Random Instruction Sequences
- ARM Conformance TestSuite
- Information Flow Analysis
- Processor Verification
- Testcase Generation
- Software Verification
- ARM Spec
- Boot OS
- Fuzzing Firmware
- Software Verification
Preparing public release of ARM v8-A specification

- Enable formal verification of software and tools
- Public release planned for 2016 Q4  2017 Q1
- Liberal license
- Cambridge University REMS group currently translating to SAIL

Talk to me about how I can help you use it
The New Bottleneck: Specifications

- Required for formal verification
- Too large to be “obviously correct”
- Reusable specs enable “virtuous cycle”
  - Increases Scope / Applicability requirements
  - Converge on correct specification

Looking for interns in Security and Correctness - contact me