Undefined, Unspecified, Non-deterministic, and Implementation Defined Behavior in Verifiable Specs

Clifford Wolf
Symbiotic EDA

8th RISC-V Workshop
Barcelona 7-10 May, 2018

Why and how to not specify something?

- It is quite usual for a specification to have “holes”, cases for which the specification does not specify a behavior.
  - If the concrete behavior is deemed irrelevant because “nobody should do that anyways”.
  - And if it might complicate implementations to restrict the range of valid behaviors in certain weird corner cases.

- But: **There are many ways not to specify something.**
  - Each spec has it’s own definitions and nomenclature for this.
  - We do not have that right now for RISC-V.
  - <u>This presentation aims at starting that conversation.</u>

- *A specification in the context of this presentation is one concrete specification, such as RV64G or RV32IC.*

Users of an ISA Specification

- Software engineers
  - Assembler programmers, Compiler writers
  - *Just don’t do anything that’s not specified*

- Software security engineers
  - *What can we expect from the hardware if we do the not specified thing anyways?*

- Hardware design engineers
  - Just do something safe and simple for anything that’s not specified
  - *But what exactly does “safe” mean?*

- Hardware verification engineers
  - For HW security reasons we need definitions for what not specified behavior can do
  - For verification purposes that definition must be falsifiable, or it is meaningless
  - In many cases (e.g. bounded model checking) it must be falsifiable with short execution traces

Taxonomy of not specified stuff

- Rest of this presentation:
  - A taxonomy of types of not specified behavior
  - Using nomenclature borrowed from different domains
    - Please don’t be angry at me for using different nomenclature than the one you are used to.

- Example used in the rest of the presentation:
  - Division by zero (Spoiler: Division by zero is fully defined in RISC-V)

Undefined Behavior

- Undefined behavior is the “just don’t do that!”– approach to not specifying behavior.
  - “software must never divide by zero”
  - “this spec is only valid for programs that do not attempt to divide by zero”
  - “division by zero is undefined”

- This usually means that the spec as a whole is void for any program as a whole if the program attempts to do the not specified thing!
- Undefined behavior is usually the default when the spec just does not mention a specified behavior.

Undefined Behavior: Can a correct implementation produce the following trace?

```
...

xor a1, a0, a0          ; a1 ← 0x12345678
mv a2, a1               ; a2 ← 0xffffffff
bgeu a2, zero, <label>  ; does not branch

...
```

Undefined Behavior: Can a correct implementation produce the following trace?

```
div t1, t0, zero


...


xor a1, a0, a0              ; a1 ← 0x12345678
mv a2, a1                   ; a2 ← 0xffffffff
bgeu a2, zero, <label>      ; does not branch


...
```

Undefined Behavior: Can a correct implementation produce the following trace?

```
div t1, t0, zero
mv t1, zero

...
```

no data dependency!

```
xor a1, a0, a0              ; a1 ← 0x12345678
mv a2, a1                   ; a2 ← 0xffffffff
bgeu a2, zero, <label>      ; does not branch


...
```

Undefined Behavior: Can a correct implementation produce the following trace?

```
...

xor a1, a0, a0            ; a1 ← 0x12345678
mv a2, a1                 ; a2 ← 0xffffffff
bgeu a2, zero, <label>    ; does not branch

...

div t1, t0, zero
```

Undefined Behavior: Can a correct implementation produce the following trace?

```
...

xor a1, a0, a0           ; a1 ← 0x12345678
mv a2, a1                ; a2 ← 0xffffffff
bgeu a2, zero, <label>   ; does not branch

...
```

Conclusion: Specs that <u>contain any</u> Undefined Behavior are extremely hard to falsify and therefore a big problem in verification.

Undefined Value

- Formally introduce a special “undefined” value that is used as the result of not specified operations.
  - Undefined values propagate to the results when used as arguments to operations.
  - Think: x-state in Verilog or VHDL

  “division by zero returns an undefined value”

  “the result of division by zero is undefined”

- When tracing a real implementation, an undefined value can be observed as any concrete value.
- Much better than undefined behavior from a security perspective.
- Still very problematic in terms of falsifiability.

Undefined Value: Can a correct implementation produce the following trace?

```
...

xor a1, a0, a0             ; a1 ← 0x12345678
mv a2, a1                  ; a2 ← 0xffffffff
bgeu a2, zero, <label>     ; does not branch

...
```

Undefined Value: Can a correct implementation produce the following trace?

```
div a0, t0, zero

...

xor a1, a0, a0                 ; a1 ← 0x12345678
mv a2, a1                      ; a2 ← 0xffffffff
bgeu a2, zero, <label>         ; does not branch

...
```

Non-deterministic Behavior Unpredictable Behavior

- Allow more than one behavior, and which behavior is used is determined at runtime in a unpredictable manner.
  - This means the same program run twice might not return the same result each time.
  - Usually used for things like ordering of concurrent events, such as concurrent memory accesses from multiple hearts.

“division by zero returns an unpredictable result”

Unspecified Value
Unspecified Behavior

- Unspecified Value: The instruction will not do anything crazy and just return a value, the spec just doesn't say which value

  "division by zero returns an unspecified value"

- Unspecified Behavior: The instruction will not do anything crazy and behave within reasonable bounds outlined somewhere in the spec. Usually a spec contains a chapter or appendix for that.
  - This might include options like not writing to the destination register.
  - The "reasonable bounds" for unspecified behavior should be specified in terms of architectural state. Otherwise it is not verifiable.
  - Handwavy bounds such as "must not compromise security" are useless!

Implementation Defined Behavior / Value

- Similar to Unspecified Value / Behavior, but the implementation must document which behavior is implemented.

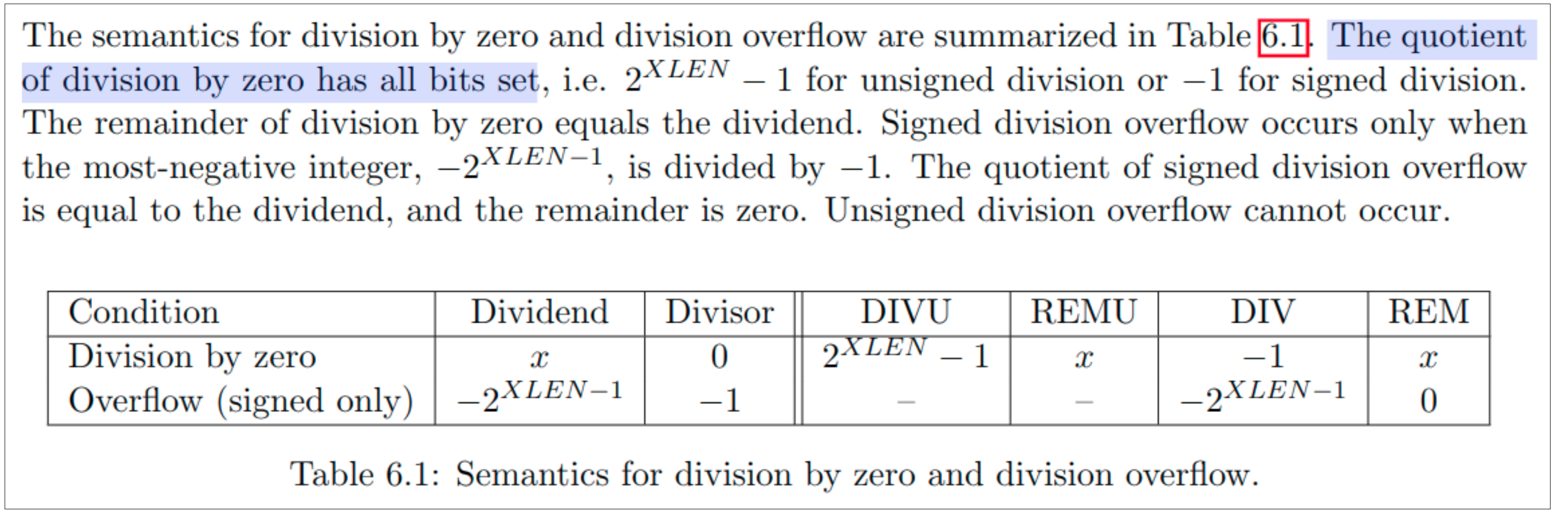
    "division by zero returns an implementation defined value"

- The spec must describe what the bounds for implementation defined behavior are

Fully Specified Behavior

- In many cases the best choice is to just specify a behavior, even in cases that are rarely hit by software.

  "division by zero returns -1 (all bits set)"



The semantics for division by zero and division overflow are summarized in Table 6.1. The quotient of division by zero has all bits set, i.e. $2^{XLEN} - 1$ for unsigned division or -1 for signed division. The remainder of division by zero equals the dividend. Signed division overflow occurs only when the most-negative integer, -2^{XLEN-1}, is divided by -1. The quotient of signed division overflow is equal to the dividend, and the remainder is zero. Unsigned division overflow cannot occur.

Condition	Dividend	Divisor	DIVU	REMU	DIV	REM
Division by zero	x	0	$2^{XLEN} - 1$	x	-1	x
Overflow (signed only)	-2^{XLEN-1}	-1	–	–	-2^{XLEN-1}	0

Table 6.1: Semantics for division by zero and division overflow.

Specification holes in RISC-V Spec

- Volume I: User-Level ISA, Version 2.2
  - Almost everything is fully specified
  - Exceptions:
    - **Non-determinism** used as necessary in memory model
    - Support for unaligned memory access is **implementation defined behavior**.
    - V-extension vector length is **implementation defined**.
    - F/D/Q-extensions used to have more implementation defined behavior, but that has been eliminated in recent revisions of the specification.

- Volume II: Privileged Architecture, Version 1.10
  - CSRs contain many optional (thus **implementation defined**) bits and features.
  - Especially in M-mode there are a couple of instances of **undefined behavior**, mostly implicit by omission of specified behavior.

Conclusion (1/2)

- Undefined Behavior, Undefined Value
  - Leads to incredibly hard to falsify specs!
  - Therefore they should not be used in specs for verifiable systems!
- Non-determinism
  - Leads to non-reproducible test results
  - Complicates verification if some cases are rare
  - Should only be used when absolutely necessary
  - For example for multi-threaded memory access

Conclusion (1/2)

- ~~Undefined Behavior, Undefined Value~~
  - ~~Leads to incredibly hard to falsify specs!~~
  - ~~Therefore they should not be used in specs for verifiable systems!~~

- **Non-determinism**
  - Leads to non-reproducible test results
  - Complicates verification if some cases are rare
  - **Should only be used when absolutely necessary**
  - For example for multi-threaded memory access

Conclusion (2/2)

- Unspecified
  - No definitive “known good” reference to compare against
  - Complicates testing and formal verification
  - Usually leads to “unofficial implementation defined behavior”

- Implementation defined
  - In many cases as good as fully specified
  - Just need to make the reference implementation parameterizable

- Fully specified behavior
  - Should be preferred whenever there is no good reason for any of the above

Conclusion (2/2)

- Unspecified
  - No definitive “known good” reference to compare against
  - Complicates testing and formal verification
  - Usually leads to “unofficial implementation defined behavior”
- **Implementation defined**
  - In many cases as good as fully specified
  - Just need to make the reference implementation parameterizable
- **Fully specified behavior**
  - Should be preferred whenever there is no good reason for any of the above