浮点数的确定性问题

不确定性说明

官方不确定性说明：

> *https://github.com/WebAssembly/design/blob/main/Nondeterminism.md*

The following is a list of the places where the WebAssembly specification currently admits nondeterminism:

- New features will be added to WebAssembly, which means different implementations will have different support for each feature. This can be detected with `has_feature`, but is still a source of differences between executions.
- When threads are added as a feature 🦄 even without shared memory, nondeterminism will be visible through the global sequence of API calls. With shared memory, the result of load operators is nondeterministic.
- Except when otherwise specified, when an arithmetic operator returns NaN, there is nondeterminism in determining the specific bits of the NaN. However, wasm does still provide the guarantee that NaN values returned from an operation will not have 1 bits in their fraction field that aren't set in any NaN values in the input operands, except for the most significant bit of the fraction field (which most operators set to 1).
- Except when otherwise specified, when an arithmetic operator with a floating point result type receives no NaN input values and produces a NaN result value, the sign bit of the NaN result value is nondeterministic.
- Fixed-width SIMD may want some flexibility 🦄
  - In SIMD.js, floating point values may or may not have subnormals flushed to zero.
  - In SIMD.js, operators ending in "Approximation" return approximations that may vary between platforms.

> *https://webassembly.org/docs/faq/*

Why is there no fast-math mode with relaxed floating point semantics?

Why is there no fast-math mode with relaxed floating point semantics?

Optimizing compilers commonly have fast-math flags which permit the compiler to relax the rules around floating point in order to optimize more aggressively. This can include assuming that NaNs or infinities don't occur, ignoring the difference between negative zero and positive zero, making algebraic manipulations which change how rounding is performed or when overflow might occur, or replacing operators with approximations that are cheaper to compute.

These optimizations effectively introduce nondeterminism; it isn't possible to determine how the code will behave without knowing the specific choices made by the optimizer. This often isn't a serious problem in native code scenarios, because all the nondeterminism is resolved by the time native code is produced. Since most hardware doesn't have floating point nondeterminism, developers have an opportunity to test the generated code, and then count on it behaving consistently for all users thereafter.

WebAssembly implementations run on the user side, so there is no opportunity for developers to test the final behavior of the code. Nondeterminism at this level could cause distributed WebAssembly programs to behave differently in different implementations, or change over time. WebAssembly does have some nondeterminism in cases where the tradeoffs warrant it, but fast-math flags are not believed to be important enough:

基本结果

原因阐述

计算机组成原理

介绍了整数之后，下一步就是讨论浮点运算，即实数之间的运算。实数是所有有理数和无理数的集合。浮点运算能够让人们处理科学应用（与金融或商业应用相对）中很大的和很小的数。浮点运算不像整数运算，它的计算结果一般是不确定的。一块芯片上的浮点计算结果也许与另一块芯片上的不同。后面将解释为什么浮点运算无法获得确定的答案，并讨论一些程序员必须了解的陷阱。

算数规则

IEEE754定义的算数规则有加减乘除、平方根、求余和比较

ANSI/IEEE 754-1985 标准定义了基本的和扩展的浮点数格式，以及一组数量有限的算术运算的规则（加、减、乘、除、平方根、求余和比较）。

非数（测量重点）

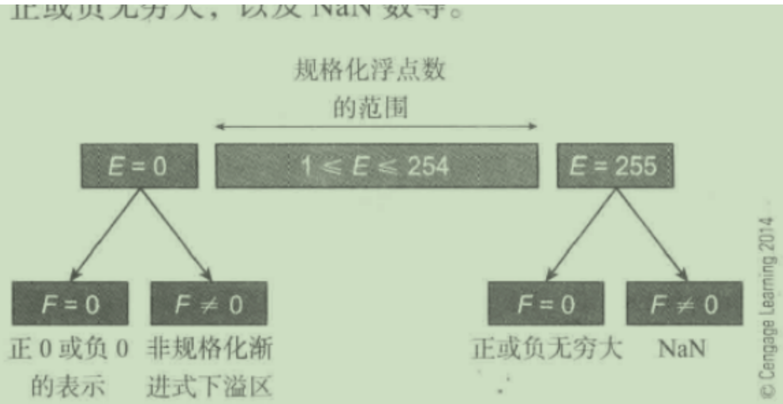
NaN的使用和定义是与系统相关的，可以用NaN来表示所需要表达的任何信息

E_max_+1表示的数字的含义？

IEEE 754 标准定义了 3 种浮点数格式：单精度、双精度和四精度（见表 2-7）。在 32 位 IEEE 754 单精度浮点数格式中，最大指数 E_{max} 为 +127，最小指数 E_{min} 为 −126，并不是我们所想的 +128 ～ −127。E_{min}−1（即 −127）用来表示浮点 0，E_{max}+1 用来表示正 / 负无穷大或 NaN 数。

非数（Not a Number, NaN）是 IEEE 754 标准的一个重要概念。NaN 是 IEEE 754 标准提供的一个专门符号，代表 IEEE 754 标准格式所不能表示的数。NaN 的使用和定义是与系统相关的，可以用 NaN 来表示所需要表达的任何信息。

小数、正或负无穷大，以及 NaN 数等。



不同的架构是否都采用截断或者舍入

实验验证

实验描述

测试不同语言编译出的wasm字节码在不同的机器上运行不同的wasm runtime是否相同。

具体测试分为三个层面

架构层面采用x86和ARM（杨硕的M1芯片）

wasm runtime 层面采用 wasmer/ wasmtime/V8进行测试

语言层面采用手写wat/Rust/C++进行测试（其中C++仅在V8上进行了测试、手写wat缺V8测试，有实验但是无法实现字节数组的打印）

C++编译环境配置

如何将C++编译为webassembly

> *https://emscripten.org/docs/getting_started/downloads.html(环境配置推荐推荐)*

> *https://developer.mozilla.org/zh-CN/docs/WebAssembly/C_to_wasm（阅读推荐）*

> *http://webassembly.org.cn/getting-started/developers-guide/*

V8Chrome测试

安装本地服务器服务，然后在Chrome中打开。

使用Nodejs的serve

```
npm install -g serve
#进入到需要打开的文件夹
serve .
#在浏览器中
http://localhost:3000
```

测试流程按照C++编译环境配置（C++ V8测试）

Rust V8测试

在rust中调用js的库，在Chrome中打开

详情见hello_wasm文件夹

手写wat 的V8测试

详情见v8write

wasmer

官方介绍文档

> *https://docs.wasmer.io/integrations/examples/instance*

使用操作流程

> *https://zhuanlan.zhihu.com/p/243210440*

Wasmtime

友情提醒：一定要好好看官网文档和文档的时间，如果是库或者包，直接看crate.io

crate.io

> *https://docs.rs/wasmtime/0.31.0/wasmtime/*

官方文档

> *https://docs.wasmtime.dev/introduction.html*

官方文档，但是比较老，里面的例子只可借鉴，不可执行

wasmer运行

cargo build --target wasm32-wasi --release

wasmer .\target\wasm32-wasi\release\floattest.wasm

wasmtime运行

wasmtime .\target\wasm32-wasi\release\floattest.wasm

浮点数和定点数的比较

如果说要使用小数，在范围允许的情况下，推荐使用定点数（效率更高）

https://zhuanlan.zhihu.com/p/149517485

架构、平台、runtime、语言

关心：wasm和架构的指令集

架构

指令集（硬件层）

平台

操作系统

runtime

rumtime和wasm可以认为是相同

Wasm官方文档

> *https://webassembly.github.io/spec/core/exec/index.html*

最准确化的描述，相对比较晦涩难懂，能用MDN文档解决的问题直接使用MDN即可：

> *https://developer.mozilla.org/zh-CN/*

比如，Wat文件格式和说明

> *https://developer.mozilla.org/zh-CN/docs/WebAssembly/Understanding_the_text_format#%E8%8E%B7%E5%8F%96%E5%92%8C%E8%AE%BE%E7%BD%AE%E5%B1%80%E9%83%A8%E5%8F%98%E9%87%8F%E5%92%8C%E5%8F%82%E6%95%B0*

语言

rust语言实现了浮点的一致性，在不包含NaN运算的情况下，wasm文件中无f32

wasm runtime

wasmer
wasmtime
wasmedge