Relaxed-math mode

Overview

- Preserve most of the performance of relaxed-simd
- Enable a path to deterministic semantics
- Most of the content here is effectively just framing
  - Framing doesn’t affect performance
  - Framing does affect the spec, long term
  - There is one thing that isn’t just framing
- Opportunity to revisit NaN bits

The general outlook

- Most instructions in relaxed-simd only need a handful of inline machine instructions to implement
  - For some instructions, nondeterminism only applies in “unlikely” situations
    - Can use inline range check + out-of-line slow path
  - Others, like min/max, can be done inline
    - Aside: encourage Intel and AMD to add IEEE 754 minimum/maximum
  - Some hosts will accept these costs in exchange for determinism
- Framing:
  - Rename opcodes from “relaxed” to “alternate”
  - Push them further out in the binary opcode space
  - Put the desired deterministic semantics in the core spec
  - Add an optional *relaxed-math mode* where they’re nondeterministic

fma

- Many cloud/edge hosts
  - Always have fma hardware
  - Have use cases that benefit from determinism
    - Wizer / snapshotting / live migration / replay / etc.
- Some hosts which need extreme determinism
  - Aren’t sensitive to floating-point performance
- Framing: Specify fma as the deterministic behavior, with mul+add as optional behavior in relaxed-math mode

What kind of nondeterminism

- In relaxed-math mode:
  - List, or used staged compilation?
  - Either way:
    - List nondeterminism would at least be contained within relaxed-math mode

bfloat16 dot product

- Non-deterministic in three different ways
  - I don’t know of a deterministic semantics we could pick that could be practical
    - But I’d be happy to be corrected
- Proposal: Remove this instruction

Revisiting NaN bits

- Wasm’s NaN bits are a surprising exception to Wasm’s overall determinism
- Proposal: Move NaN bits nondeterminism to relaxed-math mode
  - ARM’s “Default NaN mode” is considered an *optimization*
  - On RISC-V, this is effectively the default mode
  - Overhead of NaN canonicalization for eg. 4x4 matrix multiply (load input and store output) if we canonicalize just at the stores: 5% (on a 2019 CPU), 9% (2015), 15% (2012)
  - IEEE 754 NaN propagation rules are a “should” not a “shall”
    - It’s intended for a debugging scheme which no popular platforms implement
    - It also gets used in R for the NA value
      - But Wasm’s existing NaN semantics already preclude this
  - Compatible with NaN boxing
  - Compatible with JS engines
  - Engines that don’t want canonicalization overhead can use relaxed-math mode

Summary

- Remove relaxed_dot_bf16x8_add_f32x4 (aka bfloat16_dot_product)
- Rename relaxed_madd (aka relaxed_fma) to alternate_fma.
- Rename relaxed_nmadd (aka relaxed_fnma) to alternate_fnma.
- Rename the remaining relaxed_* to alternate_*.
- Change the binary opcodes for alternate_* (after the 0xfd prefix) to start at 0x100000 instead of 0x100.
- Define deterministic semantics for all alternate_* instructions:
  - For swizzle, laneselect, q15mulr_s, and dot_i8x16_i7x16_*, define them to have some agreeable deterministic semantics TBD.
  - For fma/fnma, define them as IEEE 754 fusedMultiplyAdd (single rounding), adjusted for the negation in fnma and for Wasm's overall exception, rounding mode, and NaN stance.
  - Define the rest to be identical to their non-alternate_ counterparts.
- Change Wasm's NaN semantics:
  - The result of any non-bitwise floating-point instruction when it returns a NaN is a canonical NaN (sign bit is zero, quiet bit is one, remaining mantissa bits are zero).
- Define a "relaxed-math mode". In this mode:
  - All the alternate_* instructions have the non-deterministic semantics proposed in the relaxed-simd proposal.
  - Wasm's NaN behavior is nondeterministic, using the NaN semantics previously specified in the core spec.