Zmmul

Multiply Without Divide

https://github.com/riscv/riscv-isa-manual/blob/master/src/m.tex

(see section "Zmmul Extension, Version 0.1")

Public Review Comments

There was a single comment about the spec that simply questioned the need for it and how it might cause tool fragmentation
(as opposed to specifics of the spec itself)

IMHO, I think it trades off implementation complexity with increasing the fragmentation to extensions,
which increase the burden on IP vender and compiler engineers.
I wonder if it is really worth to add this extension and splitting mini-extensions out from current extension?

Dividers can have different implementations, each with different trade-offs, providing many choices for architects.
Those who even don't want to introduce even a single adder in some small ASIC designs can still use the GCD algorithm
while sharing ALU as adder. This won't increase the implementation difficulty or area cost (maybe only add a FSM).
For FPGA users, they also can leverage the DSP block inside FPGA
(for example: https://www.xilinx.com/support/documentation/ip_documentation/div_gen/v5_1/pg151-div-gen.pdf)
So I don't think divider provides difficulty or burden to small ASIC core or FPGA implementations.

There was a discussion about trap&emulate for div, but further spec clarification made it clear that isn't allowed.

My answers were 2-fold:
- We shouldn't need to dictate that FPGA implementations require FPGAs that have DSP blocks
- Gate cost is not the only cost; design verification for a block that completely changes a simple fixed pipeline to a variable one
  (especially with concurrent interrupt events, returning loads and stores, etc) will add cost to the implementation.

The other answer was that there is a possibility of fragmentation, but that will be filtered by the mandates of profile definitions, e.g.

RVA profiles (for apps processors) will mandate the M extension, and so software in this space can remain ignorant of Zmmul's existence.
The RVM profiles might(?) declare that M is supported-optional whereas Zmmul is unsupported-optional,
so standard library vendors shouldn't feel obligated to support Zmmul by itself.
In effect, this would restrict Zmmul's applicability to those who are willing to support their own software stacks.