The landscape of RISC-V floating point support with BF16 at the centre

Kenneth C. Rovers

CPU Architecture, Imagination Technologies

Abstract

There is a bewildering amount of number formats, particularly floating point ones. Each ISA will need to consider which to support. For RISC-V, we argue we need to consider supporting all, but ratify only a few; the ones that have become mainstream. Current ratified extensions of the RISC-V ISA suffices, with the exception of BF16 (Brain floating point format) support (and perhaps rising star FP8). Supporting BF16, however, is also not straightforward. We will identify several use cases; as a storage format, for efficient computations, and as an arithmetic format, and expose BF16 is not a standard, highlighting differences in implementations. For RISC-V support we need to balance flexibility with (hardware) efficiency and will recommend complementary extensions with the base supporting flushing subnormal numbers and round-to-zero.

Introduction

Since the rise of machine learning a myriad of number formats have been introduced aiming to improve the efficiency of training and inference. These in addition to the many historic number formats in existence, as well as number formats from other domains. Does RISC-V need to support the full landscape to be successful? On the integer side life simply consists of power of two in 2's complement form (and perhaps fixed-point and binary coded decimal). On the floating point side things become more interesting. The RISC-V floating point special interest group (FP SIG [1]) has created a catalogue of known formats, extracting commonalities and identifying differences. This in turn raises interesting questions in how to support this diversity. We will quickly explore this landscape and summarise the finding in the next section.

We will find we're lacking support for the Brain floating point format (bFloat16/BF16), which is an increasingly important number format for AI applications, where it was realised that often you need the range of single precision floating point numbers but not the precision. BF16 is however not a standard format, not in the IEEE sense nor as a ratified RISC-V ISA extension. It does not even seem to be intended as a standard but rather as an internal optimisation. Yet there many implementations, including RISC-V implementations. Imagination Technologies as well has implemented support for BF16 in its real-time embedded CPU, the RTXM-2200, and is planning support for BF16 in its upcoming products. Again, we will explore the landscape, this time of use-cases and implementations.

Finally, we will discuss our journey so far and suggest directions for the road ahead.

Method and result

The view

As mentioned, a catalogue of (floating point) formats is created by the FP SIG [1]. This list is by no means exhaustive but has already identified over 50 formats.

We can roughly classify these formats. The majority are based on the well-known IEEE 754 standard [2] with the formats consisting of a fixed-size logarithmic part (exponent) and a fixed-size linear part (fraction). Then there are tapered floating point formats aimed at increasing accuracy in the range near 1 and which have a variable width exponent and fraction [3], and specialised formats such as the LNS (logarithmic number system) format which has no linear part [4].

It is clear that IEEE-based formats are the primary candidate to support, but even for IEEE compliant numbers we have base 2 and base 10 variants, and there are already over ten IEEE-like formats focussed on AI applications.

Everyone can choose to support their favourite format as a custom extension, however, so we argue we only need to support the mainstream formats as ratified extensions. The task then becomes to identify those that will become mainstream.

Already defined by the RISC-V ISA are half (Zfh), single (F), double (D), and quad (Q) precision, with single and double precision clearly the most ubiquitous floating point formats in use. Decimal floating point is reserved but not yet defined.

That roughly matches the support of other ISAs with one notable exception; BF16. That arguably makes BF16 a mainstream format (although it is perhaps too early to tell). Additionally, there is an IEEE standardisation effort on the way for FP8 (8-bit IEEE-like), so we recommend to closely follow and prepare for supporting that.

BF16

The BF16 number format is developed by the Google Brain research team for deep learning applications. It is similar to single precision (F32) but with a smaller fraction, i.e. it is an IEEE-like format with a sign bit, 8 exponent bits, and 7 mantissa bits (instead of 23 in F32), meaning it has the same range but less precision. This smaller fraction significantly reduces the hardware cost of implementing arithmetic. A BF16 FMA, for example, is less than a quarter of the area of a F32 FMA. It was found that for AI applications the accuracy of F32 wasn't needed and the area was better spend on increasing the number of operations per second achieved by using BF16.

That isn't the only use case however. We can identify three use cases for BF16:

1. As a storage format.

Memory bandwidth requirements can have a significant impact of performance. By storing data in BF16 format memory bandwidth is halved while the rest of the computation maintains accuracy. This only requires instructions for converting to and from BF16, or no hardware support at all if truncating and zero-extending.

2. For efficient matrix computations

By performing the multiplications in matrix operations with BF16 inputs but accumulating in F32 most of the accuracy is retained while also achieving most of the area (and bandwidth) savings. Typically, this requires a mixed-precision (widening) FMA instruction.

3. As an arithmetic format

The full algorithm is now performed in BF16 for even further efficiency improvements. Generally this is supported by implementing all the same instructions other floating point formats provide.

The most common use case is 2), supported for example by PyTorch or TensorFlow. There is also a use-case for 3) however. Both CUDA [5] and StableHLO/XLA [6] support BF16 as an arithmetic format. Internal analysis has shown a 20 times smaller error rate than half precession numbers on some benchmarks giving results with less than 0.2% error rate differences against F32.

It is recommended that these use cases are all supported as complementary extensions as currently planned.

ISA support

While most ISAs support BF16, it is not a standardised number format, i.e. it is not fully IEEE-compliant. Therefore implementations can differ. The differences seen are if subnormal numbers are supported or treated as zero (flushed), and what rounding modes are supported or used. As the format was developed by Google arguably their implementation could be considered the standard, and as we will see most ISAs follow their choices.

Table 1 lists the instruction set, subnormal support, and rounding mode for the Google TPU [7], Intel AVX-512_BF16 [8], Armv8.2-A [9], Armv8.2-A extended BF16 support [9], and Nvidia Ampere tensor cores [10].

The Google and Intel implementations support conversions and FMA instructions, the others also support other matrix supporting operations like dot-products. Note that Google, Intel, and ARM flush subnormal numbers, but ARM has an option to make flushing selectable, while Nvidia always supports subnormals. There does not seem to be a clear consensus on the used rounding mode.

ISA	Instr.	Subnormal	Rounding
Google	CNV, FMA	flush	?
Intel	CNV, FMA	flush	RNE
ARM	CNV, FMA, MAT	flush	sel cnv, RTO
ARM_E	CNV, FMA, MAT	sel	sel
Nvidia	CNV, FMA, MAT	support	RTZ

Table 1: BF16 ISA support

In general, we argue it is most compliant to flush subnormals and support RTZ rounding as the default. The lack of standardisation will require extension options enabling other choices however.

Discussion

In the RISC-V community we have an opportunity to carefully consider a well-designed and clean ISA. This requires considering all potential number formats to support. As will have become clear however, the valuable opcode space required for full support would be large. It is likely that wider standardised opcodes will be used eventually naturally allowing common and most often used instructions and number formats in narrower encodings and uncommon ones in wider encodings. The alternative would be using modes, i.e. CPU state, allowing e.g. the BF16 instructions to use the same encoding as half precision but in a different mode. The advantage is avoiding wider instructions and thus larger binaries, but the disadvantage is less efficient execution when having to change mode.

The BF16 use cases all have their place, care should therefore be taken in ensuring they are consistent. While the lack of standardisation is unfortunate and may require some configurability it should be noted that BF16 primarily is meant to improve area and efficiency hence such options should be judged in that light, favouring flushing and RTZ.

References

[1] https://lists.riscv.org/g/sig-fp.

[2] IEEE Computer Society. "IEEE Std 754™-2008".

[3] R. Morris. "Tapered Floating Point".

[4] S.Lee, A.Edgar. "Add. "The Focus Number System"".

[5] CUDA Toolkit Doc. "Bfloat16 Precision Intrinsics"

[6] OpenXLA Project. "StableHLO Specification".

[7] S.Wang, P.Kanwar. "BFloat16: The secret to high performance on Cloud TPUs".

[8] Intel. "BFLOAT16 Hardware Numerics Definitions".

[9] "Arm® Architecture Reference Manual, Armv8-A"

[10] Fasi et al. "Numerical behavior of NVIDIA tensor Cores"