An exploration of Inexact FP Arithmetic

The intent is to carry IN THE FP DATUM an Inexactnes indication, which (i. a.) would permit distinguishing overflow from infinity, and underflow from zero. Conceivably a mode bit would provide a re-interpretation of the four basic rounding modes as well as details of the arithmetic and interpretation of the formats.

The basic idea is to dedicate the low-order significand bit to the I-bit, or inexactness indicator, thus reducing the effective precision by one bit. Special considerations are needed for DFP, where unnormalised (but normal magnitude) representations are by definition exact: these are representations with a zero high-order digit and a nonzero biased exponent (i.e. not subnormal). The I-bit is a regular low-order bit for these "unnormal" DFP numbers, so that in particular exact integers and exact fixed-point decimals can be represented with their preferred quantum (i.e. expected exponent). For normalised or subnormal DFP representations the low-order bit is however an I-bit, so that the effective low-order digit is even. (This means we have to be especially careful in defining the rounding rules in the to-nearest family, i.e. not the directed rounding modes.) Luckily the low-order bit is independent of BID vs DPD encoding, so the rules to be defined here do not depend on the encoding.

The above only applies to nonzero finite numbers. The largest magnitude is one ulp less than for regular FP, and the smallest magnitude is twice that of regular FP for a given format. The smallest normal magnitude is the same as for regular FP (because its low-order bits are all zero).

Inexact Infinity (the result of overflow rounded away from zero) can use the normal I-bit in the case of DFP, but in the case of BFP it takes one bit away from the NaNcode. Given the bit-reversed interpretation used by IBM (e.g. when converting NaNcodes between BFP and DFP), this is a loss of the high-order bit, so NaNcodes interpreted as small integers (i.e. the kind that is expected to survive format conversions) are not affected.

Exact and Inexact zero present the biggest conceptual problems, because of the interpretation of the sign bit. Regular 754 arithmetic has precise rules for signed zero. These are of course not affected, as IFP is to be a distinct mode of FP arithmetic. The question is how these rules are to be transferred to the IFP mode.

The problem is that there are two kinds of inexact zero: with or without a definite sign. The square of an inexact nonzero number is definitely positive, and multiplying it by a definitely negative number (not inexact zero) should yield a definitely negative result, even in the presence of underflow. On the other hand, the difference between two inexact numbers with identical representations is an inexact zero with indeterminate sign.

One possible solution to this conundrum is to model IFP arithmetic on the projective (one-point) compactification of the Reals, with an unsigned exact Infinity, namely the inverse of an unsigned exact zero. We can then re-interpret the sign bit of Infinity and Zero to denote an indeterminate sign when the I-bit is zero. (When the I-bit is one, Infinity denotes a signed overflow, and Zero denotes a signed underflow.)

IFP arithmetic with finite nonzero numbers

The rounding is described in terms of the Parity (for round-to-nearest-even only), Rounding and Sticky bits (PRS), following conceptually exact arithmetic with unbounded precision and range. Given that the effective precision is one bit less than for regular FP arithmetic with a given format, the position of the PRS bits is shifted one bit to the left, i.e. the R bit occupies the position of the eventual I-bit (lowest-order significand bit). For DFP the P-bit needs to be described separately as 10 is not divisible by 4, but the main focus is on the RS bits anyway.

As a first step, the S bit is initialised to the OR of the operands' I-bits. Note that for "unnormal" DFP, there is an implied I-bit of 0 because the complete significand (with a zero high-order digit) is presumed exact.

Except for "unnormal" DFP, the low-order significand bit of each operand is set to zero, and the conceptually exact intermediate result is computed. In the case of division or square root, infinitely many low-order bits may occur, but they will collectively simply contribute to the S-bit.

At that point, all bits or digits to the right of the R-bit are ORed together into the already-initialised S-bit. This works for DFP because the R-bit is the low-order bit of a low-order result digit, so that only complete digits are to its right. (In this context, significance decreases going right.)

For DFP, if the S bit is now zero (typically operands were "unnormal") and the result is such that the high-order digit would be zero ("unnormal" result), the R-bit is ignored, i.e. the complete low-order digit survives even if it is odd (there is no I-bit in "unnormal" DFP numbers). The result is indeed exact and no rounding applies. If however the result has more than precision-1 digits (or if it was an inexact square root or quotient), it is subject to normalization.

For BFP, and for DFP when the S-bit is one, the intermediate result is subject to normalisation, which determines the actual position of the R-bit and hence the set of bits to its right that will contribute to the S-bit. This conceptual shift is accompanied by an exponent adjustment, provided the biased exponent remains positive. If this is not possible, the result will be subnormal, and sufficiently many zero high-order bits (or DFP digits) will be tacked on at the left end to lead to a biased exponent of zero. If this leads to all zero digits or bits to the left of the R-bit, we have total underflow, and the result will be an inexact zero or inexact 2*Dmin (smallest subnormal nonzero magnitude), depending on the rounding direction. There is one exception: subtraction of two equal magnitudes: this tricky case will be discussed later. (If overflow or underflow is trapped, the exponent range is effectively unbounded and does not affect rounding or inexactness; the trap handler will be provided with a wrapped exponent or a separate scale factor, as appropriate, and normalization is always feasible if indicated.)

If exponent overflow occurs, the S-bit is forced to 1, and the result will be inexact Infinity or Nmax, depending on the rounding direction.

Normal rounding is then applied, and the R-bit is replaced with the S-bit, which becomes the result's I-bit. Rounding overflow could lead to delayed exponent overflow, which is treated as described above. Note that rounding overflow cannot turn an "unnormal" DFP into a normal number, because only exact results can be "unnormal", in which case there is no rounding.

Round-to-nearest-even for DFP needs a separate description. That's because only odd low-order digits are present (the low-order bit is occupied by the I-bit, which is one), though they actually represent even digits (the I-bit overlays a conceptually zero bit). Again, "unnormal" numbers (which don't have an I-bit) are not affected since they are always exact, and rounding simply does not apply.

Tie breaking is needed when R=1 and S=0. The idea of ties-to-even is to break ties in a balanced manner. This is not possible when there are only five digits to choose from in the low-order digit. One possibility would be to use the parity of the next-to-lowest order digit to decide whether to round up or down, in a balanced manner. Not much more complicated, and closer to the BFP model, would be to break ties towards a multiple of four, where the P bit is the XOR of the low-order bit of the next-to-last digit and the next-to-last bit of the last digit. Note that DFP often uses the financial ties-away-from-zero rule, which rounds away from zero when R=1, disregarding both P and S bits, so the choice of rule for ties-to-even may not be critical (but it has to be decided).

Round-to-prepare-for-shorter-precision is actually not needed anymore since Inexactness is not lost. It can therefore be implemented as any other rounding mode, perhaps simply truncating (i.e. ignoring the R bit). This works for BFP as well as for DFP.

IFP subtraction of equal magnitudes

When both operands are exact (S=0), the result is an exact zero. For regular FP arithmetic this would be +0 except for Floor rounding. IFP uses an unsigned exact zero, so it would be +0 in all cases. For DFP the preferred quantum rule would be followed.

When at least one operand is inexact (S=1), rounding applies. Directed rounding is mostly easy: round towards zero would yield an *exact* zero (which is one of the exceptions to inexactness-preservation), whereas Floor and Ceiling yield an appropriately-signed inexact operand ulp (if only one operand was inexact or two ulps (if both were inexact). There is however a problem with round-away-from-zero: the result should be of the same magnitude as for Floor or Ceiling, but with an indeterminate sign: this rounding mode (which exists for IBM DFP) should perhaps not be supported, or this case should lead to Invalid Operation with a NaN result as a last resort. (In other words, underflow away from zero has a scale that reflects the source magnitude, like DFP zeros.)

Note: by defining signed inexact tiny rounded away from zero to be at least 4*Dmin in all cases, including when the cancelling operands are both inexact subnormals, we avoid Dmin trouble when we use tow low-oder bits to encode inexact zeros. (See discussion in the "More to do..." section.)

Round-to-nearest (note that there can be no tie, as R=0, so the various flavours of tie breaking are all equivalent) naturally leads to an inexact zero with indeterminate sign, a datum we do wish to support. Since exact zero is unsigned in IFP, we could pick the encoding of -0. For DFP the quantum is probably best computed the same way as for exact zero, since this preserves an indication of the absolute scale of inexactness. See below for the rules of arithmetic with indeterminate-sign inexact zero.

(An inexact signed zero can only arise from multiplication or division underflow, as an inexact signed difference is at least one ulp.)

Addition/Subtraction of zeros and finite numbers

If, of two zero operands, at least one is of indeterminate sign, the result is an inexact zero of indeterminate sign, except in the case of round-towards-zero, where the result is an exact zero. The same applies to the addition of inexact zeros of opposite signs, or the subtraction of inexact zeros of the same sign. The sum or difference of two exact zeros is of course an exact zero; the sum or difference of an exact zero and a signed inexact zero depends on the rounding mode, and is either an exact zero (if rounding towards zero) or a signed inexact zero of the appropriate sign (if rounding away from zero, or to nearest); there is no sign ambiguity.

If one operand is non-zero, Inexactness is propagated. An exact zero causes the result to be the other operand (with its sign reversed if it was the subtrahend), and the rounding mode has no effect. For an inexact zero (signed or not) and round-to-nearest (any flavour), the result is the other operand (with sign reversed if subtrahend), and forced Inexact. For a signed inexact zero and directed rounding, the result is either of the same magnitude, or rounded up or down, depending on whether the sign of zero and the rounding direction agree or disagree. For an inexact zero of indeterminate sign and directed rounding, the result is always rounded up or down based on the rounding direction alone.

Multiplication of finite numbers by zeros

Multiplication by exact zero yields exact zero, regardless of rounding mode. Multiplication by inexact zero leads to inexact zero of the same kind (signed or indeterminate-sign); signed zero has the exclusive-OR of the operand signs. Division of zero by nonzero follows the same rule as multiplication.

Division of finite numbers by zeros

Division of exact zero by any inexact zero yields exact zero, regardless of rounding rule. Division of any nonzero by exact zero yields exact unsigned infinity (and may signal divide-by-zero). Division of nonzero by inexact signed zero behaves like overflow and rounds to inexact infinity or to inexact Nmax, depending on the rounding direction like overflow; the sign is the exclusive-OR of the operand signs. Division by inexact zero of indeterminate sign leads to infinity of indeterminate sign, regardless of rounding mode.

This is the usual source of infinities (other than literals).

Square Root of zero

Square root of negative signed zero, or of zero of indeterminate sign, signals Invalid Operation and results in a NaN. Square root of exact zero is exact zero; square root of positive inexact zero is positive inexact zero except when rounding towards zero or Floor, in which case it is exact zero.

Infinities

As seen under "division by zero" above, there are four infinities, which are the inverses of the four zeros: exact unsigned, inexact signed, and inexact of indeterminate sign. Since exact +Inf is considered to be unsigned we could use -Inf to encode infinity of indeterminate sign. The signed inexact infinities have the I-bit set, and for BFP these would look like NaN in regular FP arithmetic, which is a shame.

It might be better to invert the meaning of the I-bit for BFP infinities, so that Inexact signed infinities behave like regular infinities when used in regular FP arithmetic, and the exact unsigned or inexact Inf of indeterminate sign would indeed be treated as NaNs, which seems appropriate; in fact, they should be made signalling NaNs. But this only applies to BFP; for DFP the I-bit is part of a payload that will be ignored by regular DFP arithmetic; it just so happens that the sign would be misleading for exact or indeterminate-sign infinities. (This issue is discussed further under "NaNs", below.)

Addition/Subtraction of infinities

If at least one operand is exact infinity, the result is exact infinity regardless of rounding direction. Even Inf-Inf = Inf (not NaN) for two exact infinities, because -Inf == Inf (unsigned) and Inf-Inf = Inf+(-Inf).

Otherwise, if at least one operand is an infinity of indeterminate sign, the result is an inexact infinity of indeterminate sign -- except for Floor and Ceiling rounding, where it is a negative or positive inexact infinity, respectively. The same applies to the addition of inexact infinities of opposite signs, or the subtraction of inexact infinities of the same sign. An exception is round-away-from-zero which, if it is supported at all, would yield exact infinity (just as round-towards-zero of underflow yields exact zero).

Multiplication by infinities

Multiplication of exact zero by inexact infinity is exact zero. All other multiplicative combinations of zeros and infinities are an Invalid Operation and result in a NaN.

Multiplication of a finite number by inexact infinity leads to inexact infinity of the same kind (signed or indeterminate-sign); signed infinity has the exclusive-OR of the operand signs, in round-to-nearest (any flavour). Round-towards-zero of a signed result, whether explicit or due to Floor or Ceiling, leads to Nmax of the appropriate sign. Round-away-from-zero of a signed result, whether explicit or due to Floor or Ceiling, leads to inexact infinity of the appropriate sign. Implicit round-away-from-zero (Floor or Ceiling) of infinity of indeterminate sign leads to signed inexact infinity of the appropriate sign. When supported, explicit round-away-from-zero leads to exact infinity.

Division of infinity by a nonzero finite number follows the same rule as multiplication.

Division by infinities

Division of exact zero by any infinity yields exact zero, regardless of rounding rule. Division of inexact zero by any infinity leads to inexact zero, of indeterminate sign if either the zero or the infinity had indeterminate sign, otherwise the sign of inexact zero is the XOR of the operand signs.

Square Root of infinity

Square root of negative signed infinity, or of infinity of indeterminate sign, is an Invalid Operation and results in a NaN. Square root of exact infinity is exact infinity; square root of positive inexact infinity is positive inexact infinity except when rounding towards zero or Floor, in which case it is positive Nmax.

Format conversions of zeros and infinities

Format conversions of unsigned zeros or infinities, whether exact or of indeterminate sign, propagate unchanged (i.e. same kind in the new format), except that exactness is forced for explicit round-towards-zero of zero, or explicit round-away-from-zero of infinity.

Format conversions of inexact signed zeros or infinities present a problem however, for directed rounding of zeros away from zero, or of infinities towards zero. The problem is that the size of Nmax and Dmin changes. So when converting to a larger exponent range, the source Nmax or 2*Dmin should be converted (with the appropriate rounding) to the target format, and when converting to a smaller exponent range, rounding should be to the target Nmax or 2*Dmin.

Text representation of Inexact FP entities

Suggestion: place '?' between the sign and the number. For ?0 or ?Inf, the sign is relevant; for 0 or Inf without '?' the sign is irrelevant and meaningless (and would not be generated), and indeterminate sign (valid only for 0 and Inf) would be indicated with '??' and no sign (on input a sign in front of ?? would be ignored).

NaNs, and encoding of Inexactness and Unsignedness

For DFP the situation is easy: NaNs don't have an I-bit, so the payload is not affected, and the distinction between Inf and Nan is independent of the payload. DFP infinities are capable of having a payload, which is simply considered non-canonical if nonzero for regular FP. DFP infinities could also encode the indeterminate-sign case by using two payload bits, so that the actual sign bit could simply be ignored.

For BFP we have to steal at least one bit from the NaNcode, because we need an I-bit for Infinity. In fact, we might as well take TWO low-order significand bits (which correspond to high-order NaNcode bits in the IBM BFP interpretation of NaNcodes, so small integers -- the preferred expression of NaNcodes -- would not be affected). This would avoid the issues mentioned earlier for encoding indeterminate-sign infinity as -Inf.

In fact, it might be nice to do the same thing for zero, but that would double again the smallest magnitude to 4*Dmin, and more seriously, break the (a==b) vs (a-b==0) equivalence. It would however greatly simplify the definition of the sign operations, which are supposed to be blind -- yet should not as a side effect change exact zeros to indeterminate-sign zeros, or vice versa.

Comparisons

(TBD -- there are some tricky issues here: are two inexact numbers of the same nominal value comparable? They could be treated either as EQUAL or UNORDERED, depending on the intent. IEEE 754 does have two flavours of comparison, quiet and noisy, so perhaps that can be exploited here.)

Application

This exploration was prompted by a paper by Prof. Siegfried Rump: "Interval Arithmetic Over Finitely Many Endpoints", Ru11c.pdf on www.ti3.tu-harburg.de (under Publications for Prof. Rump), as well as recent discussions on stds-1788@listserv.ieee.org (P1788 workgroup).

The idea of distinguishing overflow/underflow from true infinity/zero popped up a number of times in stds-1788 -- notably in late Oct 2008, and in April and Sept 2010, prior to resurrection in Dec 2011.

The arithmetic proposed here is intended for Interval Arithmetic, and is well-defined only for directed rounding modes Floor and Ceiling. It must of course be defined for the other rounding modes as well, but inconsistencies can occur if different rounding modes are applied in incompatible sequences, for example by a sequence of additions with rounding alternating between Floor and Ceiling.

Prof. Rump's proposed arithmetic differs from what is being discussed here primarily by supporting only signed inexact infinities and zeros (the former called HUGE and the latter called TINY, each representing an interval from Nmax to Inf, or from Zero to Dmin, respectively. The finitely-many endpoints are themselves sets of reals, in fact either intervals or singletons, satisfying certain constraints. The paper is mathematical and addresses coding issues only indirectly.

More to do...

The encoding of indeterminate-sign zero and infinity needs to be revisited, most likely by stealing two low-order significand bits for Zero and Inf, and checking out the consequences, the most damaging of which is what happens to Dmin, the smallest non-zero (subnormal) magnitude. Since this is done only for zero, there is no problem with DFP, as coefficient values from 0 to 9 (we would use 0, 1, 3) are encoded identically in BFP, DPD and BID. (The term "coefficient" denotes the significand interpreted as an integer, not as a fraction.) The great benefit of this approach is that the sign bit of unsigned or indeterminate-sign zeros and infinities simply becomes irrelevant, so that the blind sign bit operations are not affected at all.

Another to-do is complete operation tables for all rounding modes and all combinations of zeros, positive and negative finites, and infinities.

Tentative analysis of iDmin = 4*Dmin (two low-order bits for Inexact Zero)

Interpretation of the 3 low-order bits, all higher-order bits being zero, i.e. we are looking at the bottom range of subnormals, which in fact look identical in BFP, DPD and BID:

000	Exact (unsigned) zero; sign bit is irrelevant
001	Inexact signed zero (corresponds to Rump's TINY)
010	Exact 2*Dmin difference of two exact subnormals (could be supported)
011	Inexact zero of indeterminate sign; sign bit is irrelevant
100	Exact 4*Dmin difference of two exact small numbers (could be normal)
101	Total underflow rounded away from zero (signed nonzero inexact)
110	Exact 6*Dmin difference of two exact small numbers (could be normal)
111	Inexact 6*Dmin difference of two small numbers (partial underflow)

What's a bit unusual here is that the smallest exact nonzero magnitude is half that of the smallest inexact nonzero magnitude -- so it's not clear which one should be called iDmin (i.e. "Dmin for IFP"). But in fact the magnitude of an inexact cancellation can be arbitrarily large, so the fact that it cannot quite be as small as in the case of the subtraction of two equal-magnitude inexact subnormals should not matter much.

This scheme is consistent with the equivalence of comparison of two numbers and the comparison of their difference with exact zero -- assuming that two inexact numbers of equal magnitude compare Unordered (as their difference would be of indeterminate sign).

We also need to consider the effects of multiplication and division. The worst effect is probably that the inverse of the smallest positive inexact is only half the size of the smallest positive exact value. In general the difference between inverses of very small numbers is wildly inaccurate, and numeric programs must already deal with such ill-conditioning.

****

Michel Hack, IBM Research, 7 Jan 2012