To : Distribution November 27, 1956

From : R. B. Lazarus

Subject: FLOATING POINT NUMBERS FOR STRETCH

Symbol : T-7

EXPONENT OVERFLOW

The following suggestions for Stretch floating point numbers are based on the fundamental assumption that all problems can be divided into two kinds, as follows:

1) Exponent overflow[1] constitutes an unforeseen disaster, and requires some form of human investigation before the calculation can profitably be continued.

2) Exponent overflow has been foreseen, but is not to be ignored; the program knows what to do about it (e.g., rescale, or restart from some previous cycle, with different parameters).

The suggestions are also based on the observation that no presently contemplated instruction can produce, from legal operands, a result whose exponent has spilled by more than one bit.

Suggestion No. 1. Pretend that the exponent has an overflow bit, or, which is entirely equivalent, that the exponent overflow detection takes place one bit shy of the exponent limit. If this were really done, then the result of an operation which brought about exponent overflow would be a perfectly correct number. Actually, the overflow bit need only be conceptual, not real, since the information is contained in the overflow trigger. That this is sufficient will become clear.

1. i.e., positive exponent overflow. Negative exponent overflow will be called "exponent underflow".

Suggestion No. 2. Let it be improper to use as an operand, except for rescaling, an overflowed number.

It now remains to show how problems of the two kinds mentioned above are properly taken care of.

For kind (1), the only thing that could possibly be desired, when exponent overflow occurs, is immediate and unconditional transfer of control to a special routine. The normal requirement on the special routine would be to print some diagnostic information and then call the next problem. If machine errors turn out, after all, to exist, it might be desired to return to a previous point and repeat. It will be observed that under no circumstances will it be useful to continue, since investigation is clearly necessary when a number of super-astronomical proportions arises unforeseen. Therefore, use of the overflowed number as an operand is meaningless.

For kind (2), it is again clear that immediate and unconditional transfer of control is required, this time to the portion of the program which knows what to do about exponent overflow. (This suggests that the address to which control is transferred unconditionally upon exponent overflow be specifiable by the program, but if the address is built into the machine then the cost need only be an extra transfer order.)

It will again be noted that overflowed numbers need never be used as operands, except in the case that the program desires to rescale and continue. But in this case only a rescaling operation is necessary, and this can be accomplished with the logical vocabulary, rather than with the floating point arithmetic vocabulary.

It will now be noted that two previously proposed features have quietly disappeared, namely the "I" tag for overflowed numbers and the

"allow", or "ignore", or "non-breakin" mode of operation. These have, of course, disappeared together, since the tag is unnecessary if one requires immediate correction of the difficulty. Responsible for this disappearance is the fundamental assumption that exponent overflow may never be ignored, and this must now be justified.

The justification for problems of kind (1) has already been given, namely that investigation is clearly necessary if such a number was unforeseen. The justification for problems of kind (2) rests partly in a frequency argument and partly in a specific questioning of the cases where ignoring would be justified. If one exponent overflow occurred per day per 704 in a problem of kind (2), its frequency, compared to all orders, would be much less than 10^{-8}. Since ignorable overflows certainly occur in practice with frequencies down by orders of magnitude, time considerations do not enter at all, but only convenience. To measure convenience, one must only ask what the conditions are under which an overflow may properly be ignored. There are two cases. The first is the case where the overflowed number will never be used. In this case it makes no difference what the results of using the number as an operand are, and there is really no cost in convenience; the special program need only transfer back into the main flow and continue. The second is the case where a number of real interest has an inverse dependence on the overflowed number, and only here can one claim added convenience from replacing the overflowed number automatically be a word having the properties of infinity. However, what is required of the special program in this foreseen case is not very much; the program must skip to the place where the zero result would be calculated and insert the zero. It is

claimed that these cases will arise with such ridiculously low frequency that the advantage of the infinity is not worth even a penny. In fact, the case is much stronger, because every extra feature on the machine and every extra sentence in the manual acquires a small negative value through the people who will learn but not use, and occasionally waste machine time through confusion or decreased attention to things of more importance[2].

It is sincerely to be hoped that these remarks will lay once and for all the ghost of infinity, for Stretch, and we shall now pass on to the more difficult question of exponent underflow[3].

---

2. It is irresistible here to insert a short sermon about the dangerous but wide-spread notion that extra flexibility should always be added to a machine or to a program when this can be done at small cost to those adding the extra flexibility. The fallacy lies in ignoring the finite nature of those using the machine or the program. One must remember that the increased flexibility always decreases the efficiency with which the machine or program is used by those for whom the extra flexibility is not relevant. For this reason, the extras must be shown to give advantages which outweigh the concomitant disadvantages. For programs, this difficulty can usually be circumvented by supplying alternative programs, without the extras, but for machines the only possibility for circumvention seems to lie in alternative machine descriptions, simplified and incomplete, and this is treacherous ground.

3. Automatic detection of numbers exceeding prescribed limits seems a very desirable Stretch feature, but is not really related to exponent overflow, and should be discussed elsewhere. The subjects seem to have become associated only because the thought occurred, and was mentioned, during a discussion of exponent overflow.

EXPONENT UNDERFLOW AND ZERO

One may again divide all problems into kinds (1) and (2), replacing "overflow" by "underflow", but it is not this time so meaningful. For kind (2), the nature of physical problems is such that a) the frequency of occurrence is much higher than for overflow, and b) in virtually all cases the thing to be done about it is to replace the overflowed number by zero. For kind (1), it will happen much more frequently that unforeseen underflows occur, and in most cases investigation will reveal that they should have been foreseen and set to zero. However, we certainly do not have a complete reversal of the situation, since "disastrous" underflows seem much more likely than ignorable overflows.

The following suggestions are not made with as great conviction as those for overflow, but it should be pointed out that they are closely related to the conclusions reached, after considerable study, in the design of Maniac II.

Being careful to remember that we are concerned now only with exponent underflow and not with zeros arising from input or cancellation, let us suggest that there be two modes of operation, one entirely analogous to that for overflow, and the other consisting of the immediate and automatic replacement of the underflowed number by a zero of some kind; for the moment, take it to be a number with zero mantissa and the maximum legal negative exponent. For problems of kind (1), the first mode is to be used. If, in the course of debugging, or when using a code for a novel set of parameters, an unforeseen underflow occurs, then the immediate situation is the same as for overflow. One cannot pretend that anything else is possible without having the problem (or at least a portion of it)

become one of kind (2). The difference is that it will certainly happen not infrequently that the investigation does transform a portion of the problem into one of kind (2), and therefore Stretch must offer reasonably convenient facilities for running a portion of a problem in the second mode. Obviously, the most convenient facility is a bit in each instruction, but it seems most unlikely that one can justify this use of an instruction bit, rather than some other use, since much cheaper methods will be reasonably convenient; e.g., a special instruction to change modes, or an instruction bit, but only on a few common orders, such as Store.

For a problem of kind (2), the second mode obviates the need of special programming for the common case of desiring zero, and the first mode is used for the rest, with appropriate special programs, as for overflow.

Now let us consider the properties of the zero mentioned above. For addition and subtraction, it has all the properties of the null operator. For multiplication, and as a dividend, it does not have those properties, since it will in general lead to a result with zero mantissa but some new exponent. This leads into the subject of the E bit tag. (The question of zero divisors is a separate one, involving the concept of the divide check.) It now seems relevant to turn attention to zeros arising from input and cancellation.

The following assumptions concerning zero seem valid:

1) Input zeros and zeros resulting from exponent underflow while in the second mode should have the properties of the null operator. The reasoning is that the programmer has decreed that these numbers should in no way contribute significant bits to the result, and therefore they had best stay completely out of the way.

2) Zeros produced by complete cancellation are in a class with exponent underflow itself, in the sense that they may be unforeseen disasters or they may be foreseen zeros.

The first assumption seems clearly to call for an E tag, which will easily enable Stretch to maintain the null operator properties. Let us then forget the zero described above, and assume the E tag.

The second assumption seems to call for two modes of operation for cancellation, quite independent of the two modes for exponent underflow. (Note that "modes of operation" is only another phrase for "trigger testing".) However, the present situation is more complicated. The main difficulty is that many problems will expect cancellation (implying zero) during the early cycles, but will want protection against unforeseen cancellations at later stages. The mode switching would be rather inconvenient.

The answer seems to lie in the fact that the "disastrous" cancellations are just limiting cases of comparably "disastrous" partial cancellations. This indicates that <u>all</u> zeros should behave as null operators <u>except</u> a few which actually constitute a <u>minority</u> subset of a larger group of numbers with the characteristic that they do not have enough significant bits. (If the emphasis on the null sounds radically contrary to the discussions of the past three weeks, it should be remembered that the situation is radically changed a) by isolating the divide checks, and b) by removing the infinities.)

The best conclusion seems to be as follows:

1) All zeros should be numbers with E tags and have the properties of the null operator, whether they are input zeros, cancellation zeros, or zeros resulting from exponent underflow while in the second mode.

2) Loss of significance from cancellation is dangerous, and facilities should be incorporated in Stretch to warn of it. However, complete cancellation is only a part of the problem; it would not seem satisfactory to warn only of this part, even if there were a convenient way to do so without interfering with intentional complete cancellation. It does seem satisfactory to give warning without immediate interruption.

Regarding (1), it must be noted that there is no way to prevent the programmer from putting in numbers with zero mantissa and no E tag. How Stretch should operate with such numbers, however, belongs with a discussion of unnormalized arithmetic, to be given elsewhere.

Regarding (2), discussions with IBM engineers seems called for.

DIVIDE CHECK

In normalized arithmetic, a divide check should occur whenever a one bit displacement of the dividend is insufficient to permit a legal division (i.e., one where the quotient of the mantissas is of magnitude less than one). This should be interpreted so as to include division by zero. Immediate and unconditional transfer to a special program is again required, since one has either a) an attempt to do unnormalized work in an improper fashion, or b) a situation fully equivalent to exponent overflow (i.e., division by zero).

Divide checks do not occur in unnormalized arithmetic, except for division by zero; this, too, belongs with the discussion of unnormalized arithmetic.

For the special case of zero over zero, present experience seems to suggest that zero is a satisfactory result. However, it would be safer,

with negligible cost in time or convenience, to call this a divide check, and let the special routine detect this special case (from the preserved operands), if desired.

Distribution:
S. Dunwell
F. Beckman
J. Griffith
L. Sarahan
D. Sweeney
B. C. Carlson
R. M. Frank
Max Goldstein
H. G. Kolsky
R. B. Lazarus
N. C. Metropolis
E. A. Voorhees
D. F. Woods
File