Modifications of System/360 Floating-Point

by

Leonard J. Harding Jr.

The University of Michigan Computing Center - August 1966

Introduction

The architectual changes advocated by this report are limited to those which correct defects that programming cannot avoid without severe degradation of the System/360. Since the computational capabilities of the larger models are currently constrained by strict compatibility, it would appear that this is an overriding consideration and hence, these changes are requested in all models.

Long Operand Addition, Subtraction and Compare

The long intermediate sum must be extended to 15 hexadecimal digits and possible carry, i.e., a guard digit must be incorporated.

Long Operand Multiplication

The long intermediate product fraction must not be truncated before postnormalization. This amounts to provision for a 15 hexadecimal digit intermediate product.

Halve Instruction

Both the short and long operand HALVE instructions must provide for postnormalization of up to one hexadecimal digit

followed by truncation to the appropriate length. Note that this requires that both instructions retain a guard digit.

Underflow and Overflow Interruptions

When an underflow interruption is taken, the result register must contain the correct sign and mantissa and a characteristic 128 larger than the true value. When an overflow interruption occurs, the result register must contain the correct sign and mantissa and a characteristic 128 smaller than its true value.

Since the internal structure and microprogram of the Models 65/67 have been made available to the University of Michigan, some attempt has been made to indicate the magnitude of these changes for these two systems. This is not meant to, nor should it be taken as implying that the proposed changes be restricted to these two models.

Long Operand Addition, Subtraction and Compare

The proposed extension of the long operand intermediate sum to 15 hexadecimal digits and possible carry only eliminates the occurrence of anomalous results when normalized long operands are used. To eliminate the idiosyncrasies that arise from the use of unnormalized operands would require either

(1) prenormalization of the operand with the larger characteristic up to the characteristic difference, or
(2) an intermediate sum consisting of 28 hexadecimal digits and possible carry.

Because the first alternative would require that the microprograms for the normalized and unnormalized operations be separated, and because of the infrequent use of unnormalized operands, the first of these must be rejected on the basis of inefficiency.* The second alternative would probably require too much additional hardware, especially in the smaller models, and would also be inefficient for normalized operands. It is only reasonable therefore, to attempt to obtain the minimal change that suffices for normalized operands. In any case, the dangers inherent in the use of unnormalized operands can be alleviated by appropriate education of the programming community. In this respect, the current inaccuracies of the long operand

*On the Model 67, checking to determine whether prenormalization is necessary would increase long operand addition time by roughly 25%.

additive operations differ from the problems involved in using unnormalized operands, since they are effectively beyond the control of the programmer.

What is the effect of providing a guard digit? For like sign additions of normalized operands, the guard digit cannot affect the value of the computed sum since postnormalization (to the left) is never required. For unlike sign additions which require no preshifting to align the hexadecimal points, the guard digit will be zero and hence, cannot affect the result. For unlike sign additions which require preshifting, however, a single guard digit is at the same time vital and sufficient. When a guard digit is retained the relative error is independent of the operands and bounded by 16^{-13}, whereas currently the relative error is directly dependent on the operands and need not be low. According to the results of a study of floating-point addition presented by Sweeney*, it is reasonable to expect that 22.66% of the additive operations performed will result in unlike sign additions with non-trivial preshifting. Further, approximately 1/5 of these operations will require postnormalization and hence, will materially benefit from the retention of a guard digit. Thus, roughly 4% of the additive operations would be affected in anywhere from the first through fourteenth digits of the result. The important point, however, is that the retention of a single

*Sweeney, D. W.; "An Analysis of Floating-Point Addition", IBM Systems Journal 4, No. 1, 31-42, 1965.

guard digit guarantees that the relative error in every addition is independent of the operands. The original question may thus be rephrased in terms of the importance of this guarantee.

Concisely stated, the incorporation of a guard digit will increase the range of problems solvable in System/360 long operand arithmetic. By implication then, there are some problems which are not solvable to the required degree of accuracy due solely to the current hardware anomalies. Although one may question the significance of this statement, it is certainly true. Unfortunately, it would appear that only experience with the current System/360 will show whether these anomalies manifest themselves sufficiently often to be bothersome. This of course, assuming that the programmer will be able to separate the hardware generated problems from those caused by the particular method or data that he is using. In consideration of the difficulties involved in tracing inaccuracies to the hardware, however, this assumption is optimistic. Further, since the errors which arise from the current facility are directly dependent upon the operands, the accuracy which a given program achieves in its results will become even more data dependent than at present. These burdensome considerations should not be placed upon the programmer, and are more properly within the scope of hardware design.

In the Model 67, the incorporation of a guard digit need

not affect the timing of the floating-point additive operations except when both recomplementation and postnormalization are necessary. Thus, these changes will increase the execution time only when the retention of the guard digit is significant. This rather favorable situation may not be true in the smaller models because of the drastic reduction in adder width and the difference in shifting techniques. Additional hardware would, of course, be preferable from the point of view of timing; however, it would appear that in the case of the Model 67, a guard digit may be made available solely through changes in the microprogram.

Long Operand Multiplication

The anomalous results currently produced by long operand multiplications result from truncation of the intermediate product to 14 hexadecimal digits before postnormalization. This means that when postnormalization is necessary, the low order digit of the result will always be zero. Specifically, if

$$A = .a_1 \ldots a_{14} \times 16^{\alpha} \qquad a_1 \neq 0,$$

$$B = .b_1 \ldots b_{14} \times 16^{\beta} \qquad b_1 \neq 0,$$

and

$$C = A \times B = .c_1 \ldots c_{14}\, c_{15} \ldots c_{28} \times 16^{\alpha+\beta},$$

then

$$fl(A \times B) = .c_2 \ldots c_{14}\, 0 \times 16^{\alpha+\beta-1} \qquad c_1 = 0$$

or

$$fl(A \times B) = .c_1 \ldots c_{13}\, c_{14} \times 16^{\alpha+\beta} \qquad c_1 \neq 0$$

where fl(A×B) denotes the result produced by System/360. The operands A and B may be assumed normalized, since floating-point multiplication always prenormalizes the operands. The first question to ask is: What is the frequency of postnormalization?

Ignoring the possibility of carries, the digit c_1 is essentially determined by the product $a_1 \times b_1$. Assuming that

a_1 and b_1 are uniformly distributed among the 15 possible hexadecimal digits, we might expect postnormalization 20% of the time. There is empirical and theoretical evidence, however, that the first significant digit is not uniformly distributed. For example, consider the set of all products of hexadecimal fractions of the form .Z × .Y, e.g., .1 × .2 = .02, .4 × .3 = .0C, .D × .F = .C3. The distribution of the leading digit of these 225 products is given in the following table.

FIRST DIGIT	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
BEFORE NORMALIZATION	45	36	27	25	19	15	15	11	10	7	5	4	3	2	1	0
AFTER NORMALIZATION	--	37	29	27	22	17	18	13	14	10	9	6	9	4	5	4

The skewness of this distribution is apparent. Indeed, after normalization the first three digits account for 41.33% of the products, while the first six account for 67.11%. Further, these lower order digits tend to reproduce themselves. Consideration of the set of 80 products of the form .Z × .Y, where Z = 1,...,6 and Y = 1,...,F, shows that 86.25% of them will have a first significance digit less than or equal to 6 after normalization.

Some theoretical basis for this skew distribution is presented in Hamming* and the references given there. On the basis of the empirical and theoretical evidence, it appears that

*Hamming, R. W.; "Numerical Methods for Scientists and Engineers", McGraw-Hill Book Company, Inc., New York, 1962.

postnormalization may be expected in roughly 50% of the multiplications.

Though the frequency of postnormalization does influence the accuracy that may be expected in solving a given problem, it does not belie the anomalies that arise in long operand multiplication. Perhaps the most dramatic effect is illustrated by the equation $1 \times X \neq X$, which is true whenever the 14th hexadecimal digit of X is non-zero. Further, there exist operands $B>C>0$ and $A>0$ such that

$$A \times B < A \times C,$$

which is a mathematical contradiction. Alternatively, there exist $E>0$ and $A>0$ so that

$$(1-E) \times A > (1+E) \times A > 1 \times A$$

Rather than dwell on these "contradictions", however, it is instructive to consider the graph of the function $fl(A \times Z)$, where A is fixed. As Z varies over the full range of possible operands the mantissas of $fl(A \times Z)$ are clearly seen to be periodic of period 15×16^{13}, so that we may restrict Z to the range .1000 0000 0000 00 to .FFFF FFFF FFFF FF.

For descriptive purposes, it is convenient to assume that A also lies in the above range. Having fixed A, there exists a unique largest Z_L such that $A \times Z_L <$ .1000 0000 0000 00. For all $Z < Z_L$ postnormalization of the product is necessary.

Accordingly, the fraction of fl(A×Z) consists of the first 13 significant digits of the product, the 14th digit always being zero. Hence, in this range multiplication is monotonic, i.e., if $Z_1 < Z_2$ then $fl(A\times Z_1) < fl(A\times Z_2)$. Unless A is .1000 0000 0000 00 or .1000 0000 0000 01, there exists a unique smallest Z_H such that $A\times Z_H >$ .1000 0000 0000 00. For all $Z > Z_H$ postnormalization is not necessary so that the function fl(A×Z) consists of the first 14 significant digits of the product. Hence, multiplication is also monotonic in this range. Now for all Z such that $Z_L < Z < Z_H$ we clearly have

$$fl(Z_L\times A) < fl(Z\times A) < fl(Z_H\times A),$$

so that multiplication is <u>monotonic</u> over the entire range under consideration. The contradiction of monotonicity referred to above must therefore arise at the endpoints of this range.

Indeed, take $A = .a_1\ldots a_{14}\times 16^{\alpha}$ (*) and allow Z to assume only the seventeen values W_{-1} = .FFFF FFFF FFFF FF×16^{β} or W_n = .1000 0000 0000 0n×$16^{\beta+1}$, n = 0,...,F. Then

$$fl(A\times W_{-1}) = .a_1\ldots a_{13}(a_{14}-1)\,16^{\alpha+\beta},\quad a_{14}\neq 0$$

or

$$fl(A\times W_{-1}) = .a_1\ldots(a_{13}-1)F\times 16^{\alpha+\beta},\quad a_{14} = 0$$

while

$$fl(A\times W_n) = .a_1\ldots(a_{13}+b_1)0\times 16^{\alpha+\beta},\quad b_1 b_2 = na_1 + a_{14}.$$

---

*Must assume that A ≠ .10000000000000×16^{α}. Under these circumstances fl(A×Z) is always Z truncated to 13 significant digits, so that fl(A×Z) is monotonic over the entire machine range.

If a_{14} is 0 or 1 then multiplication is monotonic across the endpoints, although for $a_{14} = 1$ it obtains erroneously that $fl(A \times W_{-1}) = fl(A \times W_0)$. For $a_{14} \geq 2$ have

$$fl(A \times W_{-1}) > fl(A \times W_n)$$

for all n such that $na_1 + a_{14} < 16$. Thus, for about 87.5% of the possible choices of A, multiplication will exhibit non-monotonic behavior.

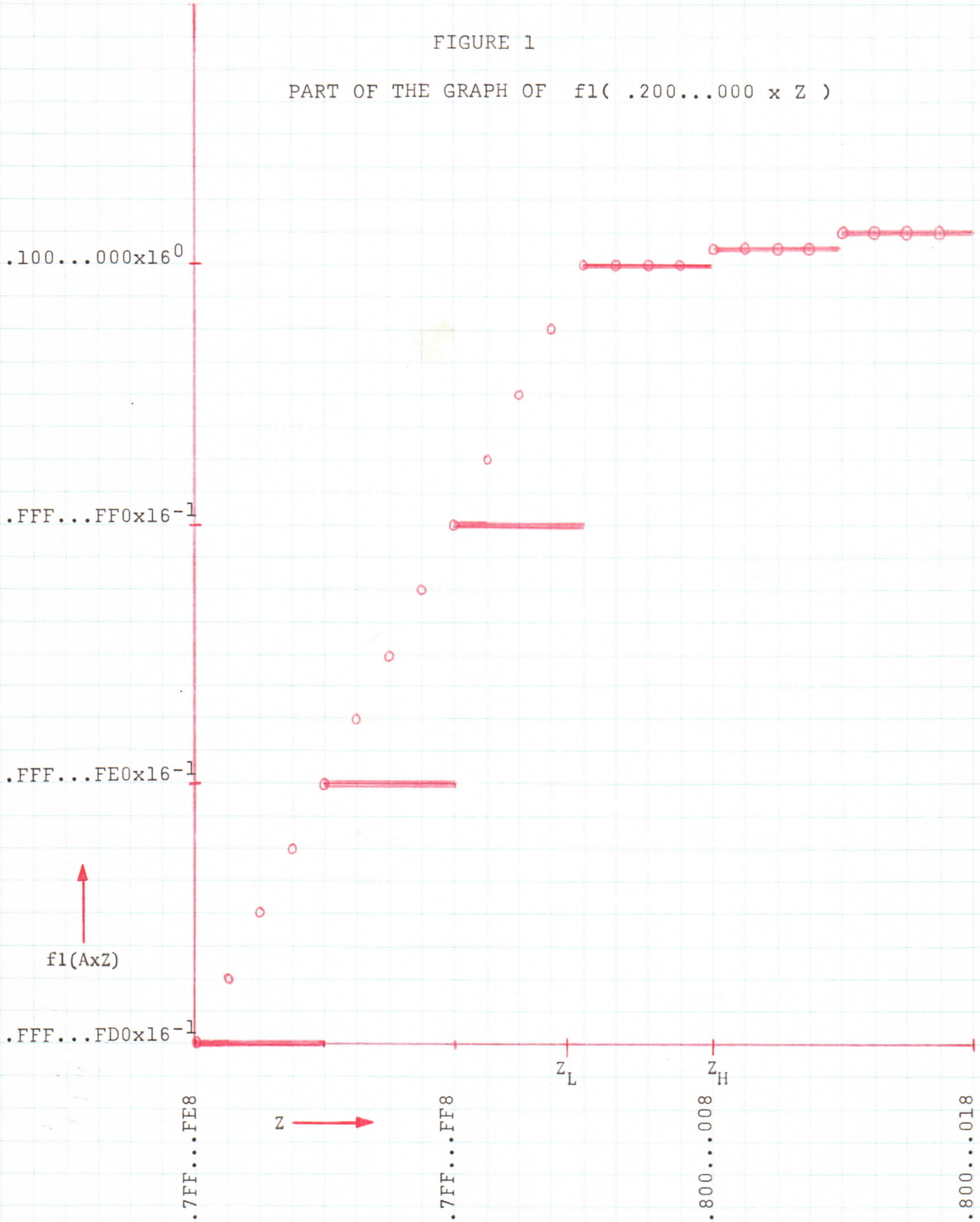
Figure 1 illustrates the typical behavior of $fl(A \times Z)$, denoted in this graph by the heavy horizontal lines. The small circles denote the result that would be obtained if postnormalization preceded truncation. The graph was constructed so as to contain both Z_L and Z_H, since the behavior of $fl(A \times Z)$ to the left of Z_L and to the right of Z_H differ radically. The important point, however, is that to the left of Z_L division and multiplication are incompatible. Indeed, the small circles also denote the result obtained by dividing the numbers on the vertical axis by .200...000. This incompatibility is a direct result of the current implementation of multiplication.

For the Model 67, the proposed change must be regarded as a minor alteration. Three ROS words in the existing microprogram must be changed, one new ROS word inserted, and the TX-trigger must be rewired so that its status may be retained through CPU clock cycles under microprogram control. The details of

the wiring necessary are well-known to IBM, since the same rewiring is necessary when extended precision is added. Since this section of the ROS program is common to both the short and long operand multiplies, the execution time of all four floating-point multiplies will be increased by 200ns. This represents about a 5% increase for the ME and MER, and less than 3% for the MD and MDR instructions.

FIGURE 1

PART OF THE GRAPH OF f1(.200...000 x Z)



The Halve Instructions

These instructions will remain virtually useless unless postnormalization followed by truncation is incorporated. Use of the unnormalized results produced by these instructions is well known to be unwise, and this situation will not be alleviated by the proposed changes for the long operand additive operations. By the same token, however, it seems reasonable that only one cycle of postnormalization be provided since this suffices if the operand is normalized. The requirement that postnormalization precede truncation is necessary for the same reason as in long operand multiplication. Indeed, if this facet of the proposal is not accepted, the incorporation of postnormalization could scarcely be justified since normalization of the truncated result can be easily, though inefficiently, programmed.

Because postnormalization implies the possibility of underflow and truncation is to follow postnormalization, it would appear that the easiest course to follow would be to terminate the halve instructions exactly as if a multiplication had been performed. In the Model 67 this would require changes in the hardware and microprogram. The hardware changes would be minimal, since they involve only the setting of triggers to control the result sign in conformity with multiplication. Three ROS words would have to be changed, and three ROS words currently in use would be freed. The execution time of both instructions would be 1.4 μsec.

Floating-Point Overflow and Underflow

Floating-point spills are a programming problem and the programmer should be able to specify any remedial action appropriate to his particular problem. Since resumption of the computation with a scaled result is often desirable, the correct sign, mantissa and a uniformly scaled characteristic must be made available. If this information is not placed in the result register at the time of the spill, it will be irretrievably lost. In the context of System/360,the reconstruction of the lost operand will be all but impossible. The hardware should not impose restrictions on the possible courses of action that the programmer may wish to adopt.

Further, the various models of the System/360 are not compatible in this respect. On any given model the result register is entirely predictable; however, different models yield different results given the same operands. It would appear that the area of incompatibility is restricted to overflows, since all models apparently yield a true zero result for underflows whether the interrupt is taken or not.

The best information available for the Model 67 indicates that all spills yield a true zero in the result register. This anomaly of the Model 67 is caused by the fact that the microprogram can only detect that a spill has occurred. No provision

has been made to allow the microprogram to distinguish between an underflow and an overflow, and in the case of an underflow, whether the interrupt will be taken. On the other hand, when a significance exception is detected, the Model 67 microprogram can interrogate the significance interrupt mask bit in the PSW.

It seems relevant also to consider what is contained in the result register prior to its being reset after the spill has been detected. For addition, subtraction and multiplication the result register is set to the result requested by this resolution prior to the detection of the spill. For division, the spill is detected before the quotient is computed. If the spill test is moved farther down in the division microprogram, the requested result would be available. At the present time, the result register contains the dividend when the spill is detected. The necessary changes in the Model 67 should prove to be minimal.

In the Model 50, overflow does not yield a zero result for addition, subtraction and multiplication. The result register appears to consist of the correct mantissa and an 8 bit characteristic. Accordingly, the sign position is always 1 since it contains the characteristic overflow. In the Model 67, the correct sign overrides the presence of the overflow bit when writing into the floating-point registers. Similar behavior of the Model 50 should not therefore be difficult.

dh