*The repertoire of* SYSTEM/360 *instructions has been expanded in the Model 85 by introducing facilities for extended-precision floating-point arithmetic. This part describes the new instructions, discusses their need, and considers the design factors that influenced their choice.*

# Structural aspects of the System/360 Model 85

# III Extensions to floating-point architecture

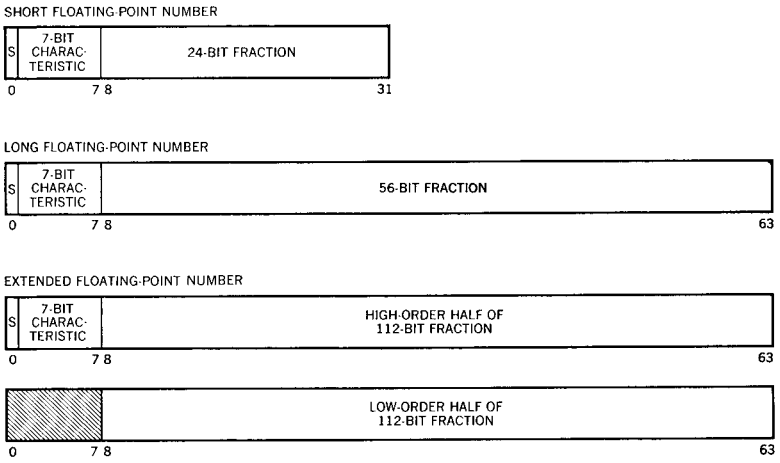
**by A. Padegs**

The floating-point facilities of the Model 85 extend the architecture of SYSTEM/360 by introducing a new floating-point number format, called extended precision, and by providing seven new floating-point instructions.

The extended-precision format, shown in Figure 1, consists of a concatenation of two long-precision formats. Hence, the extended-precision format has a fraction of 28 hexadecimal digits or 112 bits, which is equivalent to approximately 34 decimal digits. When specified as an operand, an extended-precision number consists of a sign, 7-bit characteristic, and the 112-bit fraction. The contents of the sign and characteristic fields of the low-order part are ignored. In a result, the signs of both parts are set to the same value and the characteristic of the low-order part is made 14 less than that of the high-order part. When the true low-order characteristic is less than zero, it is expressed modulo 128, but no exponent underflow is recognized unless the high-order characteristic underflows.

Table 1 lists the new floating-point instructions with their mnemonics, formats, and operation codes. Three of the new instructions provide for adding, subtracting, and multiplying two extended-precision operands to yield an extended-precision result. Two instructions provide for multiplying two long-precision operands to yield an exact extended-precision result. The two rounding instructions provide for rounding from the extended format to the long format and from the long format to the short one.

**Figure 1 Floating-point formats**




This part of the paper describes the types of applications for which the new floating-point instructions were developed, the underlying philosophy used in selecting these facilities, and some of the design alternatives considered.

## Applications requiring extended precision

The instructions contained in the original SYSTEM/360 floating-point feature provide for truncated short- and long-precision arithmetic, where, as a rule, operations with short operands yield short results, and operations with long operands yield long results. The only exception is a short-precision multiplication yielding an exact product of two short-precision operands. By clearing the low-order half of a register, a short operand can be extended to the long format, and by using the long-precision instructions on such extended short operands, long results may be obtained. However, no comparable facilities exist for long-precision arithmetic, and SYSTEM/360 float-

**Table 1 Extended-precision and rounding instructions**

| *Name* | *Mnemonic* | *Type* | *Op Code* |
|---|---|---|---|
| ADD NORMALIZED (extended operands, extended result) | AXR | RR | 36 |
| SUBTRACT NORMALIZED (extended operands, extended result) | SXR | RR | 37 |
| MULTIPLY (extended operands, extended result) | MXR | RR | 26 |
| MULTIPLY (long operands, extended result) | MXDR | RR | 27 |
| MULTIPLY (long operands, extended result) | MXD | RX | 67 |
| LOAD ROUNDED (extended to long) | LRDR | RR | 25 |
| LOAD ROUNDED (long to short) | LRER | RR | 35 |

ing-point facilities do not lend themselves to programming operations with fractions having more than the 14 hexadecimal digits provided by long-precision arithmetic.

crucial computations

Although the fraction size provided by long-precision arithmetic is ample for the great majority of present applications and those of the foreseeable future, a number of applications exist where higher accuracy is desirable. One typical example of such an application is the execution of the function $C = \sum (A_i * B_i)$ encountered in matrix multiplication. If the operands $A$ and $B$ are in long precision, the accuracy of the matrix operation can be significantly increased by developing an exact product having 28 hexadecimal digits and then performing summation on these products. This function requires facilities that, first of all, permit developing a 28-hexadecimal-digit product of two long operands, and, second, provide for forming a 28-hexadecimal-digit sum of two such operands. If one of the operands in such an instruction for addition could be specified to be in long precision, the instruction would be useful also for accumulating long operands into an extended sum.

The need for floating-point arithmetic with precision in excess of 14 hexadecimal digits exists whenever the result of a computation must be a long-precision number with all digits significant. A typical example of such an application is the evaluation of mathematical functions, such as sine, cosine, or square root. The FORTRAN subroutines that evaluate these functions in short precision contain occasional long-precision instructions to provide the required short-precision significance. Analogously, the corresponding long-precision subroutines must perform certain steps in the computations with more than 14 hexadecimal digits of significance, and extended-precision arithmetic is needed. In these types of applications, a few extended-precision instructions are inserted at crucial points in a long-precision computation, and the extended-precision facility is needed to support long-precision arithmetic. Result precision in excess of 14 hexadecimal digits is not desired.

program check-out

Another example where extended precision is needed for support of long precision is program check-out. When one suspects that, in long precision, the accumulated error due to truncation and subtraction may have reduced the result significance below the minimum number of significant digits, some assurance of the number of significant digits left may be obtained by executing the computation in a higher precision for certain selected input data and comparing the results with those obtained in long precision. In such an application, the whole task is programmed and computed in the extended-precision format, and an extended-precision result is obtained. However, the higher result precision is not a goal in itself. Once the validity of the long-precision program has been established, the extended-precision version is not needed. Extended precision in this case serves as a substitute for rigorous error analysis.

Arithmetic using a larger fraction also reduces the need for rounded arithmetic. In the absence of facilities for such higher-precision arithmetic, rounding may be needed to reduce the magnitude

of the error, and a rigorous error analysis may require rounded arithmetic as a means of making certain error-analysis techniques applicable. For these reasons, use of extended precision may serve as a substitute for rounding.

higher precision

Finally, facilities for arithmetic of higher precision are needed whenever the significance provided by long-precision arithmetic is insufficient. This situation may arise because the initial operands have more than 14 hexadecimal digits of significance, because higher result significance is desired, or because in long precision the accumulated error introduced by truncation and subtraction reduces the result significance below an acceptable level. As machines become faster and more reliable, longer computations are attempted. But as the number of operations performed on a piece of data increase, so does the accumulated truncation error. Higher precision, therefore, has to be used in certain stages of the computation to maintain the same result significance.

### Requirements for extended precision

To provide a significant increase in the size of the fraction and to permit operations involving an exact product of two long-precision numbers, the arithmetic of the next-higher precision must provide for fractions of at least 28 hexadecimal digits or 112 bits. The format having 28 hexadecimal fraction digits is referred to as the extended format, and the associated arithmetic is referred to as extended-precision arithmetic. However, to provide for the few applications requiring even larger fractions which occur, for example, in integer arithmetic, the facilities should preferably be open-ended, so as to permit one to program arithmetic on operands obtained by concatenating three or more long-precision fractions. Arithmetic yielding precision three or more times higher than long precision will henceforth be referred to as multiple-precision arithmetic.

The requirements for extended-precision floating-point facilities, therefore, are as follows:

- The fraction must have at least 28 hexadecimal digits.
- The number range must not be less than that in short or long precision.
- The format must be compatible with the present format, namely, it must be possible to extend a long-precision number to extended precision by appending zeros, and, conversely, it should be possible to truncate an extended-precision number to long precision by discarding the low-order digits.
- The data format and the set of instructions must be optimized for extended precision, but should also facilitate multiple-precision floating-point arithmetic.

To make the floating-point instruction set attractive for operations on numbers having more than 14 hexadecimal digits of fraction, two additional facilities are essential: first, instructions for

adding and subtracting are needed that shift the fraction across the boundary between the long-precision format and its extension, and, second, it must be possible to obtain the exact product of two long-precision numbers. Once these facilities are available, it becomes feasible to program all other operations.

## Design considerations

Since the justification for extended-precision arithmetic comes primarily from a few specialized applications and since it is not anticipated that extended precision will become, in the near future, the basic format of floating-point arithmetic for general use, the full SYSTEM/360 floating-point instruction set is not made available for extended-precision arithmetic. Instead, the extended-precision facilities were chosen with the underlying philosophy of including only those instructions that provide a new function or that provide a function that has both a significant use and is not readily performed by means of the present SYSTEM/360 instructions. In view of this, instructions only for adding, subtracting, and multiplying are provided.

addition

Three types of instructions can be considered for adding extended-precision operands:

AXDD: $X \leftarrow D + D$
AXXD: $X \leftarrow X + D$
AX: $X \leftarrow X + X$

where $D$ and $X$ represent operands in the long and extended format, respectively. Analogous alternatives exist for subtraction. The first of the three instructions (AXDD) is the simplest one to implement and provides the most elementary tool for adding numbers with fraction sizes in excess of 14 hexadecimal digits. The second instruction (AXXD) considerably eases programming of extended-precision arithmetic by simplifying the handling of fraction overflow. But both approaches share the shortcoming that to add two extended-precision numbers, the operations require explicit addressing of the low-order part of the operand. Therefore, these instructions require an extended-precision format where the low-order part is a valid long-precision number with the sign the same as and the characteristic 14 less than that of the high-order part.

Such an approach causes the low-order characteristic to underflow whenever the high-order characteristic is less than 14, and therefore effectively restricts the range to numbers with characteristics between 14 and 127. Taking an interruption after the underflow, with the characteristic expressed modulo 128, and processing the exception as a special case, would have the effect of making the execution time for addition and subtraction prohibitively long. Alternatively, a format could be adopted where the characteristic of the low-order part of an extended-precision number is set equal to the high-order characteristic. One could then consider a set of instructions, AXXD and AXXD′, where the latter implicitly specifies

that the true characteristic of the second operand is 14 less than specified in the format. Such an approach, however, is undesirable because of its indirectness in providing the desired facility.

To avoid the need for explicit addressing of the low-order part of an extended-precision operand and the associated problems of range restriction, instructions for adding and subtracting of the type AX were adopted: AXR and SXR. They provide, at some additional cost of implementation, a direct way of adding and subtracting two extended-precision numbers and make the structure of extended-precision instructions consistent with that of short- and long-precision instructions. Although the instructions are optimized for extended precision, they make addition and subtraction of multiple-precision operands feasible. To limit the size of the error introduced when the operation involves taking the difference between the absolute values of two operands, the preshifted operand is extended with a four-bit guard digit that participates in the operation.

multiplication

The other requirement, an exact product of two long-precision numbers, is satisfied by providing a pair of multiply instructions, MXDR and MXD, that yield an extended-precision product of a long multiplicand and a long multiplier. These instructions are relatively easy to implement, and, in conjunction with the instructions for extended-precision addition and subtraction, they enable one to program the multiplication of two extended-precision numbers. Although the algorithm for the latter operation is simple, programming of the routine is complicated again by the need for scaling to avoid exponent underflow within the normal SYSTEM/360 floating-point range. Thus, normally, this operation would have to be provided by a library routine, incurring thereby the overhead penalty of subroutine calls. In view of the need for fast multiplication, the wide use of the function, and the performance gain that can be achieved by performing the operation in hardware, the extended-precision instruction set also includes the instruction for extended-precision multiplication, MXR. Although the instruction MXR can be used to perform the functions of MXDR and MXD, the need for the operation performed by MXD was deemed to be sufficient to justify the inclusion of all three instructions.

A number of alternatives can be considered for facilitating higher precision in division, such as providing a remainder in long precision or dividing two extended-precision numbers to yield an extended-precision quotient. A program to yield a correct truncated extended-precision quotient from two extended-precision operands is very time consuming. However, by making approximations that sacrifice the significance of one or more of the low-order quotient digits, the function can be evaluated in a software routine in a period of time that is comparable to the time it takes to perform the exact division by hardware. Since division is less common than addition, subtraction, and multiplication, and since a comparable alternative exists, it was decided that the cost of the hardware implementation of the function is not justified.

instruction format

It was recognized that the RX formats of the instructions for extended-precision addition, subtraction, and multiplication would eliminate a substantial amount of loading and storing, since the register space with extended-precision operands and results becomes rather limited. Each extended-precision operand requires a pair of registers, and, in the absence of the RX-format instructions, one of these register pairs has to be used for buffering the storage operand, thus reducing the effective number of registers from four to one. However, the RX-format instructions do not provide a new arithmetic function; they primarily reduce the amount of coding required rather than the execution time of the program. Since the execution of extended-precision instructions requires a relatively long time, the time for storing and fetching is not significant and may be partially overlapped with the execution time. Furthermore, implementation of the RX-format instructions would require storage references for two double words and thus introduce a new concept in operand fetching. Such change is particularly undesirable in view of the removal of the original SYSTEM/360 restrictions on the boundary alignment of operands. To comply with the ground rules of putting emphasis on function and providing only the essential instructions for programming of extended-precision arithmetic, the instructions designating extended operands are provided only in the RR format. Similarly, extended-precision instructions for loading and storing are not provided, since a pair of the corresponding long-precision instructions provide the function.

data format

The extended-precision format is defined as a concatenation of two long-precision formats, with the signs of both parts of the result being set to the same value, and the characteristic of the low-order part of the result being made 14 less than that of the high-order part. The definition of extended-precision arithmetic is such that the low-order sign and characteristic field is redundant and instead could have been used for two additional digits of fraction. However, by keeping the extended-precision fraction a multiple of the long-precision fraction, implementation of extended-precision operations is greatly facilitated in parallel machines and the execution of division in software is made feasible. Such an approach also makes the instruction set more suitable for multiple-precision arithmetic.

When the low-order characteristic underflows, it is expressed modulo 128. Since extended-precision instructions do not explicitly refer to the low-order part of an operand, underflow in the low-order characteristic does not cause recognition of exponent underflow; exponent underflow is recognized only when the high-order characteristic of the result is less than zero. The low-order characteristic and sign of an extended-precision operand are ignored, and the correct values are inserted in results only so as to make each component of an extended-precision result a valid and meaningful number by itself. The latter is convenient for such applications as programming of multiple-precision arithmetic.

rounding

To satisfy the need for rounding in floating-point arithmetic, two instructions for explicit rounding are provided—LRDR for

rounding from the extended to the long format and LRER for rounding from the long to the short format. These instructions provide an alternative to truncation when the result of a computation is shortened to the next smaller format.


## ACKNOWLEDGMENT

A number of people have participated in defining the extensions to the floating-point architecture of SYSTEM/360. Particularly significant contributions were made by J. P. Benkard, G. B. Hedrick, T. A. Kircher, and P. H. Sterbenz. The advice of V. Kahan of the University of Toronto has been very valuable.


## GENERAL REFERENCES


1. G. M. Amdahl, G. A. Blaauw, and F. P. Brooks, Jr., "Architecture of the IBM SYSTEM/360," *IBM Journal of Research and Development* **8,** 87–101 (April 1964).
2. G. M. Amdahl, "The structure of SYSTEM/360, Part III— Processing unit design considerations," *IBM Systems Journal* **3,** No. 2, 144–164 (1964).
3. D. W. Sweeney, "An analysis of floating-point addition," *IBM Systems Journal* **4,** No. 1, 31–42 (1965).