IBM System/370

Mathematical Assists

Publication Number
SA22-7094-1

File Number
S370-01

Second Edition (December 1984)

This edition obsoletes the previous edition, SA22-7094-0. It describes several new mathematical-function facilities. Changes or additions to the text or illustrations are indicated by a vertical line in the margin to the left of the change.

Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM equipment, refer to the latest *IBM System/370 and 4300 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

PREFACE

This publication describes a number of facilities which provide instructions to improve performance in certain mathematical computations. The description is in three parts. The first part describes the multiply-and-add facility. The second part covers the square-root facility. The third part describes facilities which evaluate certain mathematical functions; they are the arctangent, common-logarithm, exponential, natural-logarithm, raise-to-power, and sine-cosine facilities.

The instructions are valid in any architectural mode (System/370, ECPS:VSE, or 370-XA) available on a model equipped with the corresponding mathematical-assist facility. The reader should be familiar with the *IBM System/370 Principles of Operation*, GA22-7000, the *IBM 4300 Processors Principles of Operation for ECPS:VSE Mode*, GA22-7070, or the *IBM System/370 Extended Architecture Principles of Operation*, SA22-7085, as appropriate, and particularly with Chapter 9, "Floating-Point Instructions," of any of those publications.

The facilities discussed in this publication are not available on every model. At the time of publication, they are available only on some models of the IBM 4341, 4361, and 4381 Processors, but all the facilities are not provided on every one of those models. Publication does not imply any intention by IBM to provide the facilities on models other than those for which they are announced. For current information concerning the availability of the facilities on any specific model, refer to the latest edition of the functional characteristics publication for the model.

Terminology

As used in this publication, a floating-point *scalar* is a single floating-point number. A floating-point *vector* is a linearly ordered collection of floating-point numbers, each number being an *element* of the vector. Consequently, a vector consists of a set of elements, each of which is a scalar.

This page is intentionally left blank.

CONTENTS

OVERVIEW

The MULTIPLY AND ADD instruction performs a combination of vector multiplication and addition operations which may replace the inner loop of common matrix computations. Its function may be described as:

A = (B * S) + C

where * indicates multiplication. B is a vector that is multiplied by the scalar S. The product is added to the vector C, and the sum replaces vector A.

The three vectors (A, B, and C) are in storage. Each consists of one or more floating-point numbers called the elements of the vector. Each vector contains the same number of elements. Scalar S is a floating-point number previously loaded into floating-point register 0. The floating-point numbers are all in normalized form and in the long format of 64 bits, except that vector C may contain unnormalized elements.

Vectors in Storage

All elements of the vectors must be located in storage on doubleword boundaries, so that their addresses are multiples of 8. Successive elements of a vector are uniformly spaced; they may be contiguous (in successive doublewords) or separated by other data. The same number of elements are processed in each vector.

The increment in bytes from the address of one vector element to the next is called the element separation. For contiguous elements, the element separation is 8. If the elements are not contiguous, the element separation must be a constant multiple of 8. The element separation for vector C is always the same as for vector A, but it may differ from the element separation for vector B.

For example, consider an N-by-N matrix that is stored in column order, the convention used for IBM System/370 FORTRAN programs. The elements of a column vector are contiguous, and the column vector has an element separation of 8. The elements of a row vector, however, are not contiguous, and a row vector has an element separation of 8N. The vector of elements along the major diagonal of the matrix has an element separation of 8(N + 1). All three vector types contain N vector elements.

Vectors A, B, and C all may be different, if their storage locations do not overlap; or any two or all three may coincide. If either of the two source-operand vectors partially overlaps the result vector in storage, the result is undefined.

Instruction Execution

The MULTIPLY AND ADD instruction performs a sequence of operations that is essentially equivalent to the execution of the following floating-point instructions on each set of corresponding vector elements:

LOAD (LD)	Load element of B
MULTIPLY (MDR)	Multiply by scalar S
ADD NORMALIZED (AD)	Add element of C
STORE (STD)	Store result in A

Arithmetically, the result is the same as if those instructions were embedded in a simple loop that also included instructions to increment the storage addresses from one vector element to the next, and an instruction to branch back until all elements have been processed.

The MULTIPLY AND ADD instruction differs from such a loop in that the recognition of an exceptional arithmetic condition (exponent overflow, exponent underflow, or significance loss) does not cause a program exception to be recognized and an interruption to occur, even though the program mask in the PSW may permit the interruption. Instead, the occurrence of such a condition causes instruction execution to be completed and a nonzero condition code to be set; the result for the current set of elements is not stored, and no more elements are processed.

The MULTIPLY AND ADD instruction also differs when the instruction encounters an unnormalized multiplication operand or a vector element that is not located on a doubleword boundary in storage; execution is completed without storing the result element, and a nonzero condition code is set.

The MULTIPLY AND ADD instruction should be followed by a BRANCH ON CONDITION instruction to test for a zero condition code, indicating normal execution. Detection of a nonzero condition code may be used to cause execution of a series of floating-point instructions, other than MULTIPLY AND ADD, which refer to the same storage addresses and reprocess the current vector elements. If an exception is then encountered, an interruption occurs, which allows the usual action to be taken. No new exception-handling programs are required. See Programming Note 2 at the end of the instruction description for an example.

The MULTIPLY AND ADD instruction may be interrupted during execution for other causes, such as access exceptions, and I/O or external interruptions. As with the COMPARE LOGICAL LONG (CLCL) and MOVE LONG (MVCL) instructions, the instruction, when reexecuted, resumes at the point of interruption.

INSTRUCTION DESCRIPTION

The inclusion of an instruction mnemonic in the description does not necessarily imply that the mnemonic is recognized by current assembler programs. A programmer wishing to use this instruction in assembler-language programs with an assembler which does not recognize the mnemonic may use an appropriate macro to assist with the translation of the instruction into machine language.

MULTIPLY AND ADD Instruction

MADS R_1,R_2 [RRE]

'B243'	/////////	R_1	R_2
0	16	24	28 31

The MULTIPLY AND ADD instruction performs the vector multiplication and addition operations:

A = (B * S) + C

where A, B, and C are three vector operands, S is a scalar, and * indicates multiplication. The vector operands are in storage, where they must be aligned on doubleword boundaries. The scalar operand and all vector elements are floating-point numbers in the long format. The scalar and the elements of operand B must be in normalized form; the elements of operand C may be in normalized or unnormalized form. The elements generated for operand A are normalized.

The scalar operand is in floating-point register 0. The vector operands in storage are specified by the contents of as many as six general registers. Three of the general registers have a fixed assignment, and any others are designated by the R_1 and R_2 fields of the instruction.

General register 1 contains a 32-bit unsigned binary integer, which represents the number of elements in each vector. General registers 2 and 3 specify the address of the first element of operands A and B, respectively.

The R_1 field, if nonzero, designates a pair of general registers, called the even R_1 register (numbered R_1) and the odd R_1 register (numbered R_1+1). The even R_1 register contains the element separation for operands A and C, and the odd R_1 register contains the element separation for operand B. The element separation for a vector is the increment in bytes from the address of one vector element in storage to the address of the next element. If the R_1 field is zero, however, the field does not designate general registers 0 and 1. Instead, the elements of all three vector operands are specified as contiguous in storage, and element separations of 8 are implied. The R_1 field must be zero or contain an even number; otherwise, a specification exception is recognized.

The R_2 field, if nonzero, designates a general register, called the R_2 register, which contains the address of operand C. If the R_2 field is zero, however, the field does not designate general register 0; instead, the address of operand C is the same as for operand A, as specified by general register 2.

The number of general-register bit positions that are used to specify an address or element separation depends on the mode of operation. The System/370 and ECPS:VSE architectural modes have only 24-bit addressing, whereas the 370-XA architectural mode offers the choice of 24-bit or 31-bit addressing. When 24-bit addressing is in effect, bit positions 8-31 are used as a 24-bit address or element separation, and the contents of bits 0-7 are ignored. When 31-bit addressing is in effect, bit positions 1-31 are used as a 31-bit address or element separation, and the contents of bit 0 are ignored.

Figures 1 and 2 illustrate the contents of the general registers for the two types of addressing, where GR1, GR2, and GR3 represent general registers 1, 2, and 3, respectively. Figure 1 shows the register layout for 24-bit addressing (System/370, ECPS:VSE, or 370-XA). Figure 2 shows the layout for 31-bit addressing (370-XA only).

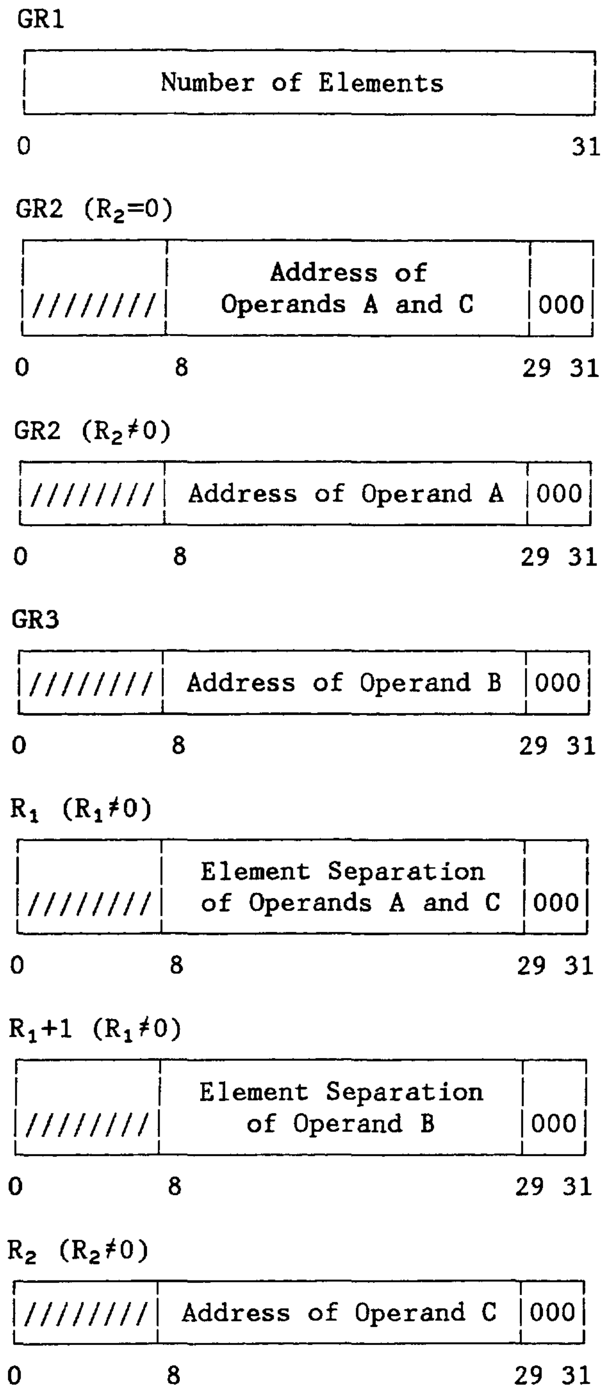


Figure 1. General-Register Assignment with 24-Bit Addressing

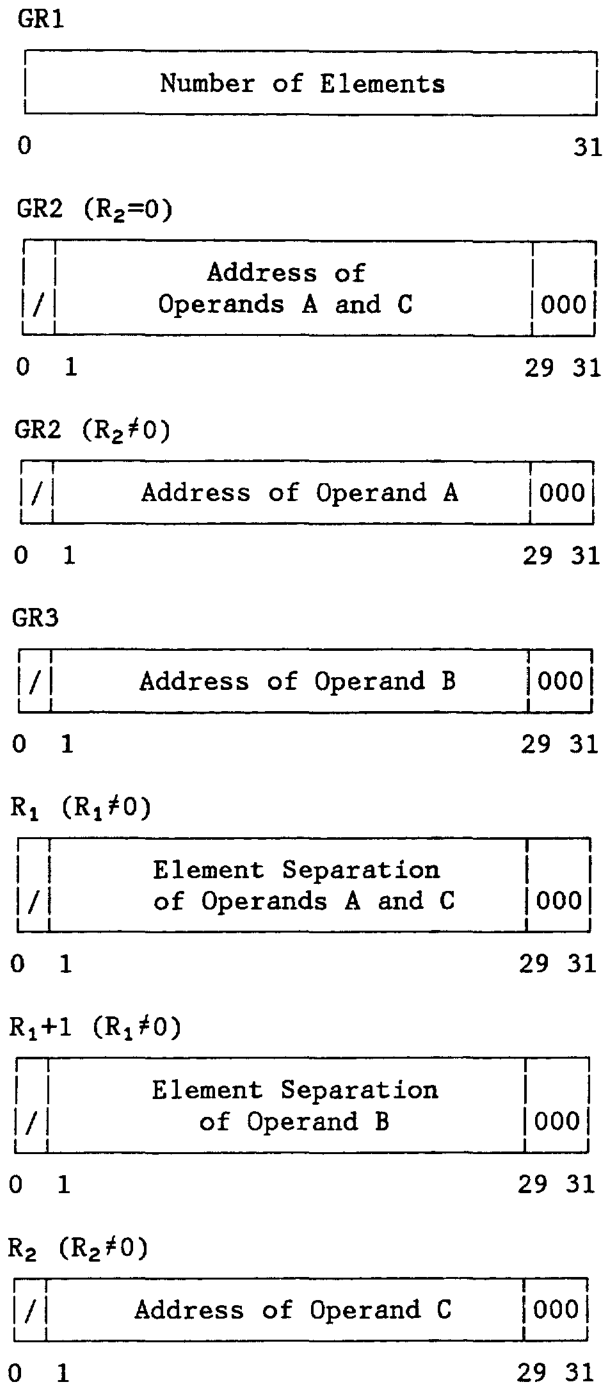


Figure 2. General-Register Assignment with 31-Bit Addressing

Figure 3 summarizes the sources of the operand addresses and the element separations, according to whether the R_1 and R_2 fields are zero or nonzero.

Field		Source of Address			Element Separation	
R_1	R_2	A	B	C	A and C	B
=0	=0	(2)	(3)	(2)	8	8
=0	≠0	(2)	(3)	(R_2)	8	8
≠0	=0	(2)	(3)	(2)	(R_1)	(R_1+1)
≠0	≠0	(2)	(3)	(R_2)	(R_1)	(R_1+1)

Explanation:

(R) Contents of general register R

Figure 3. Operand Addresses and Element Separations

Execution of the instruction begins with three tests performed in the following order: First, if general register 1 contains zero, condition code 0 is set. Next, if the fraction of the operand in floating-point register 0 is nonzero but unnormalized (the leftmost hexadecimal digit is zero), condition code 2 is set. Finally, bits 29-31 of all general registers that contain vector-operand addresses and element separations are tested for zeros; if any of those bits is one, condition code 3 is set.

If any one of the preceding tests sets the condition code, the remaining tests are omitted, instruction execution is completed, and register and storage contents remain unchanged. Otherwise, the operation proceeds by repeating the following steps until instruction execution is completed or interrupted:

1. If the fraction of the element at the address of operand B is nonzero but unnormalized (the leftmost hexadecimal digit is zero), condition code 2 is set, and instruction execution is completed. Otherwise, instruction execution continues.

2. If instruction execution continues, the element of operand B is multiplied by the contents of floating-point register 0. If either number to be multiplied has a zero fraction, the product is set to a true zero. The product and the element at the address of operand C are then added. Register and storage contents remain unchanged during this step.

3. If an exponent-overflow condition occurs during either the multiplication or the addition, it is not treated as a program exception. Instead, condition code 1 is set, and instruction execution is completed.

   If an exponent-underflow condition occurs during either the multiplication or the addition, or if the addition produces a zero result fraction, the condition is not treated as a program exception. Instead, instruction execution depends on the settings of the exponent-underflow and significance masks in the PSW:

   - When the mask that corresponds to the recognized condition is one, condition code 1 is set, and instruction execution is completed.

   - When the exponent-underflow mask is zero, exponent underflow during the multiplication causes a true zero, instead of the product, to be added to the operand-C element. Exponent underflow during the addition causes a true zero to replace the result of the addition, and instruction execution continues.

   - When the significance mask is zero and the addition produces a zero result fraction, a true zero replaces the result, and instruction execution continues.

Exponent overflow or exponent underflow during the multiplication is recognized even if the addition would bring the result back into the representable range.

Exponent overflow or exponent underflow during the multiplication is not recognized if the product in step 2 is set to a true zero.

4. If instruction execution continues, the result element is stored at the location specified by general register 2. Then, if R_1 is zero, 8 is added to the contents of general registers 2 and 3; if R_1 is zero and R_2 is not zero, 8 is also added to the contents of the R_2 register. If R_1 is not zero, the contents of the even and odd R_1 registers are added to the contents of general registers 2 and 3, respectively; if both R_1 and R_2 are not zero, the contents of the even R_1 register are also added to the contents of the R_2 register.

5. The contents of general register 1 are decremented by one. If the result is zero, condition code 0 is set, and instruction execution is completed.

If the scalar factor in floating-point register 0 is zero, it depends on the model whether an unnormalized element of operand B is recognized and condition code 2 is set in step 1, or whether instruction execution instead proceeds to step 2 and sets the product in step 2 to zero.

In step 2, the multiplication operation is the same as for the floating-point instruction MULTIPLY (MD), and the addition operation is the same as for ADD NORMALIZED (AD). Only the operand sources, the result target, and the handling of exception conditions differ.

During the additions in step 4, carries out of bit position 8 for 24-bit addressing and carries out of bit position 1 for 31-bit addressing are ignored; bit positions 0-7 or bit position 0, respectively, of the updated general registers are set to zeros.

Floating-point register 0, the element-separation registers, and the elements in storage for operands B and C remain unchanged.

Execution of the instruction is interruptible by any interruption condition for which the CPU is enabled, other than an exponent-overflow, exponent-underflow, or significance exception. A unit of operation consists of one or more repetitions of the preceding five steps, with any interruption occurring at the end of step 5 or when the instruction is completed.

When an interruption occurs during execution and the interruption condition is not one that causes termination, general register 1 indicates the number of elements remaining to be processed. The operand addresses have been updated to indicate the next set of elements to be processed, and the condition code is unpredictable.

Access exceptions for operands may be recognized for storage locations other than the locations containing the current vector elements. For each operand, however, access exceptions are not recognized for more than one element beyond the current element or for element locations beyond the last element specified.

If any of the three tests made at the start of instruction execution sets the condition code, so that instruction execution is completed immediately, then no access exceptions are recognized for any operand, the change bits for operand A are unaffected, and no PER event for general-register alteration or storage alteration is indicated.

The storage location of operand A may coincide with the location of operand B or C, if the same first-element addresses and the same element separations are specified. If both conditions are not satisfied and partial overlap occurs between the location of operand A and the location of operand B or C, the contents of the location of operand A are undefined. Those contents are also undefined if $R_1 = 2$ or, for $R_2 \neq 0$, if:

$R_2 < 4$, or
$R_2 = R_1$, or
$R_2 = R_1+1$.

Resulting Condition Code:

0	All elements processed
1	Exponent overflow, exponent underflow, or significance loss
2	Unnormalized scalar or operand-B element
3	Element address or separation not multiple of 8

Program Exceptions:

- Access (fetch, operands B and C; store, operand A)
- Operation (if the multiply-and-add facility is not installed)
- Specification

Programming Notes

1. Unlike the scalar floating-point instructions MULTIPLY and ADD NORMALIZED, the MULTIPLY AND ADD instruction requires operands to be aligned on doubleword boundaries in storage. Moreover, unlike scalar MULTIPLY, it does not multiply *unnormalized* operands. The MULTIPLY AND ADD instruction also does not cause a program interruption when an arithmetic-exception condition is recognized for which the CPU is enabled. In all these cases, the instruction sets a nonzero condition code so that scalar floating-point instructions may be used to perform the arithmetic.

   If execution of the MULTIPLY AND ADD instruction sets a nonzero condition code, fewer than the specified number of vector elements were processed. General register 1 contains the number of elements remaining to be processed in each vector. The general registers containing addresses designate the set of elements which caused the exceptional condition. The program may then execute scalar floating-point instructions in an attempt to process these elements.

   If, during the execution of these scalar instructions, an exponent-overflow, exponent-underflow, or significance exception is recognized, and the corresponding program interruption is allowed, the interruption may invoke standard fixup routines for these causes. If an unnormalized operand is encountered or a vector in storage is unaligned, the scalar instructions can process the elements.

2. The following example in assembler language illustrates the type of programming that is recommended. By adding linkage and initialization instructions, a library subroutine is created that may be called from a high-level language, such as FORTRAN.

```
L1 MADS 4,6       MULTIPLY AND ADD
   BC   8,L2      Test condition code 0
* Do following instructions if CC not 0
   LD   2,0(0,3)  Load element of B
   MDR  2,0       Multiply by S
   AD   2,0(0,6)  Add element of C
   STD  2,0(0,2)  Store element of A
   LA   2,0(4,2)  Update address of A
   LA   3,0(5,3)  Update address of B
   LA   6,0(4,6)  Update address of C
   BCT  1,L1      Branch if not done
* End of equivalent instructions
L2 EQU  *         Continue
```

In this example, the R_1 and R_2 fields of the MULTIPLY AND ADD instruction designate general registers 4 and 6. Thus, the number of elements is in general register 1; the addresses for operands A, B, and C are in general registers 2, 3, and 6; and the element separations are in general registers 4 (for operands A and C) and 5 (for operand B). The MULTIPLY AND ADD instruction is followed by a BRANCH ON CONDITION instruction and by a loop containing equivalent scalar floating-point instructions, which are executed only when a nonzero condition code occurs. The loop uses floating-point register 2 as a working register.

Note that this loop performs the same operations as one iteration of the preceding MULTIPLY AND ADD instruction, except:

- The storage operands may be unaligned, and the element separations need not be a multiple of 8.

- The multiplication operands may be in unnormalized form.

- Exponent overflow causes an interruption; and exponent underflow and significance loss cause an interruption, if permitted by the program mask in the PSW.

- A floating-point working register is needed.

- The loop needs no initial test for a zero number of elements in general register 1, because the MULTIPLY AND ADD instruction sets condition code 0 for this case.

For the special instances in which vectors C and A coincide or in which successive vector elements are in successive doublewords, the routine may be simplified accordingly.

3. Any two or all three vectors may coincide. The vectors coincide if both their addresses and their element separations are the same. Partial overlap in the storage areas for B and C may occur if neither vector overlaps with the result vector A. Partial overlap with the result vector, however, has unpredictable effects, which may differ from one model to another or from one execution to another. No check is made for partial overlap with the result vector.

   If vector C coincides with vector B, the same address should be loaded into general register 3 and into the R_2 register. The R_2 field should not designate general register 3; otherwise, it is unpredictable whether register 3 is updated once or twice for each set of elements processed. Similarly, the R_2 field should not designate general register 2 if vector C coincides with vector A. Thus, an R_2 field containing 2 may or may not have the same effect as an R_2 field containing 0.

4. Register and storage contents remain unchanged when instruction execution

ends before any elements have been processed successfully.

5. See the section "Interruptible Instructions" in Chapter 5, "Program Execution," of the appropriate Principles of Operation publication for more information concerning interruptible instructions. Also, see the programming notes at the end of the section "Program-Event Recording" in Chapter 4, "Control," of that publication regarding redundant PER events that may occur when an interruptible instruction is resumed after an interruption.

6. Special precautions must be taken if MULTIPLY AND ADD is made the target of EXECUTE. See the programming note concerning interruptible instructions under EXECUTE in Chapter 7, "General Instructions," of the appropriate Principles of Operation publication.

7. The MULTIPLY AND ADD instruction is not equivalent to the loop containing scalar floating-point instructions in the above example if accessing an operand in storage causes an addressing or protection exception. If such an exception is recognized, execution of the MULTIPLY AND ADD instruction may be terminated such that the results left in the storage areas and in the general registers are unpredictable, and the condition code may not indicate an exceptional arithmetic condition for the elements being processed. Therefore, the program must not rely on the results and attempt to resume execution after an addressing or protection exception. (See the section "Termination" in Chapter 5, "Program Execution," of the appropriate Principles of Operation publication.)

This page is intentionally left blank.

SQUARE-ROOT FACILITY

The square-root facility consists of the SQUARE ROOT instruction and the square-root exception. The instruction, which extracts the square root of a floating-point operand in either the long or short format, is four bytes long and uses the RRE instruction format. The source operand resides in a floating-point register, and the result is placed in a floating-point register. When a positive, nonzero source operand is encountered which is unnormalized, it is first normalized at the start of the operation, but without changing the contents of the source-operand location. Nonzero results are always normalized. A zero result is made a true zero.

The numeric result of the square-root operation on a valid operand is determined by the rules of arithmetic. If the result can be represented exactly in the specified floating-point format, the exact result is produced. If the result cannot be represented exactly, it is rounded to the nearest number that is representable in the specified floating-point format.

An operand that is less than zero is invalid. If the SQUARE ROOT instruction is executed with an invalid operand, a square-root exception is recognized.

Square-Root Exception

A square-root exception is recognized when the second operand of SQUARE ROOT is less than zero.

The operation is suppressed.

The instruction-length code is 2.

The square-root exception is indicated by a program-interruption code of 001D hex (or 009D hex if a concurrent PER event is indicated).

INSTRUCTION DESCRIPTION

The inclusion of instruction mnemonics in the description does not necessarily imply that the mnemonics are recognized by current assembler programs. A programmer wishing to use this instruction in assembler-language programs with an assembler which does not recognize the mnemonics may use appropriate macros to assist with the translation of the instruction into machine language.

SQUARE ROOT Instruction

SQDR R_1,R_2 [RRE, Long Operands]

'B244'	////////	R_1	R_2
0	16	24	28 31

SQER R_1,R_2 [RRE, Short Operands]

'B245'	////////	R_1	R_2
0	16	24	28 31

The normalized and rounded square root of the second operand is placed in the first-operand location.

When the fraction of the second operand is zero, the sign and characteristic of the second operand are ignored, and the operation is completed by placing a true zero in the first-operand location.

When the second operand is less than zero, a square-root exception is recognized.

When the second operand is normalized and greater than zero, the characteristic, fraction, and sign of the result are produced as follows:

- The result characteristic is one-half of the sum of the operand characteristic and either 64, if the operand characteristic is even, or 65, if it is odd.

- If the operand characteristic is odd, the operand fraction is shifted right one digit position, the rightmost digit entering the guard-digit position.

- An intermediate-result fraction is produced by computing without rounding the square root of the operand fraction, after any right shift as described. The intermediate-result fraction consists of the 15 most significant hexadecimal digits of the square-root result in the long format, or seven in the short format, where both formats include a guard digit on the right.

- A one is added to the leftmost bit of the guard digit of the intermediate result, any carry is propagated to the left, and the guard digit is dropped to produce the result fraction.

- The result sign is made plus.

When the second operand is unnormalized and greater than zero, the operand is first normalized. The operation then proceeds as for normalized operands.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6. Otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Operation (if the square-root facility is not installed)
- Specification
- Square root

Programming Notes

1. The use of the SQUARE ROOT instruction with short operands (SQER) is illustrated by the examples in Figure 4.

Operand (hex)	Decimal Value	Result (hex)	Decimal Value
42 190000	25.0	41 500000	5.0
40 400000	0.250	40 800000	0.50
40 800000	0.50	40 B504F3	0.7071...
41 800000	8.0	41 2D413D	2.8284...

Figure 4. Square-Root Examples

2. The result fraction is correctly normalized without any further left or right shifts of the intermediate-result fraction and without any further exponent adjustment. Rounding cannot cause a carry out of the leftmost digit.

3. Although a characteristic greater than 127 or less than zero may temporarily be generated during the operation, the result characteristic is always within the representable range, and no exponent overflow or underflow occurs.

   Specifically, the smallest nonzero operand in the long format consists of a one bit, preceded on the left by 63 zeros. This operand is an unnormalized number with a value of 16^{-78}, and its square root is 16^{-39}. The normalized representation of this result has a characteristic of 26 (decimal). Similarly, the square root of the largest representable operand has a characteristic of 96 (decimal). The instruction, therefore, cannot produce a nonzero result with a characteristic outside the range of 26 to 96.

The mathematical-function facilities include instructions which perform some of the mathematical operations that are commonly included as basic functions in higher-level programming languages. The instructions are ARCTANGENT, COMMON LOGARITHM, COSINE, EXPONENTIAL, NATURAL LOGARITHM, RAISE TO POWER, and SINE. They operate on floating-point numbers in either the long or short format.

Each instruction is four bytes long and uses the RRE format. The source operand is in a floating-point register, and the result is placed in a floating-point register. When a nonzero source operand is encountered which is unnormalized, it is first normalized at the start of the operation, but without changing the contents of the source-operand location. Nonzero results are always normalized. A zero result is made a true zero.

When a mathematical-function instruction performs a floating-point operation in the short format, the rightmost 32 bits of the source register are ignored as the operand is fetched. When the instruction places a short-format result in the target register, the rightmost 32 bit positions of the register remain unchanged.

The specific result produced by one of these instructions for a particular source operand or operand pair either equals or approximates the mathematically exact result. When it is necessary to approximate the exact result, the instruction returns one of the two normalized floating-point numbers which are the nearest neighbors of the infinitely precise result; which of the two numbers is returned depends on the algorithm employed in the implementation. Thus, the maximum error is less than one unit in the last place of the normalized result.

Arithmetic exceptions such as exponent overflow, operands outside the range implemented for a particular function, and other exceptional conditions do not cause a program interruption but are indicated by the condition code. If the operation is performed successfully, the instruction sets condition code 0; otherwise, the instruction a nonzero condition code, as described for each instruction, and the result location remains unchanged.

Different models may implement a different selection of these instructions, or none at all. The operand range covered for a given instruction may not always be the same from one model to another. When an instruction operand is outside the operand range implemented by the model on which the instruction is executed, condition code 3 is set. Refer to the functional-characteristics manual of a particular model for the selection of instructions and their operand ranges.

To avoid model-dependent operation, a program using one of the mathematical-function instructions should be accompanied by a subroutine which performs the entire function without relying on that instruction. Each use of the instruction should always be followed by a conditional branch to test the condition code. If the code is not zero, indicating an exceptional condition such as operands that are outside the implemented range, a branch to the subroutine may then be used to complete the operation or to handle the exception appropriately.

Programming Notes

1. The instructions evaluate the following mathematical functions:

r = arctan(x)	for ARCTANGENT,
r = log(x)	for COMMON LOGARITHM,
r = cos(x)	for COSINE,
r = exp(x)	for EXPONENTIAL,
r = ln(x)	for NATURAL LOGARITHM,
r = y**x	for RAISE TO POWER,
r = sin(x)	for SINE,

where operand x is in the register designated as the second-operand location, operand y is in the register designated as the first-operand location, and ** indicates exponentiation. The result, r, is placed in the first-operand location, replacing the operand y, if any.

2. NATURAL LOGARITHM and EXPONENTIAL perform inverse operations; that is, within operand ranges appropriate to each function, they approximate the relations:

   x = ln(exp(x)) and x = exp(ln(x))

   Similarly, COMMON LOGARITHM and RAISE TO POWER with a base (first operand) of 10 perform inverse operations which approximate the relations:

   x = log(10**x) and x = 10**(log(x))

3. The result of using RAISE TO POWER to evaluate y**0.5 for a positive y may differ slightly from the result of applying SQUARE ROOT to the same operand in the same floating-point format. Likewise, the result of using RAISE TO POWER to evaluate y**(±n), where y is positive and n is an integer, may not be the same as the result obtained by using *MULTIPLY* and *DIVIDE* instructions.

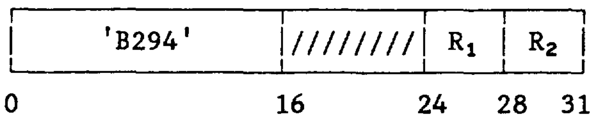
Mathematical Constants

In the instruction descriptions, *e* refers to the mathematical constant with the approximate value 2.718281828... and *pi* refers to the mathematical constant with the approximate value 3.141592653...

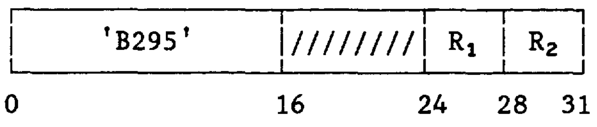
INSTRUCTION DESCRIPTIONS

ARCTANGENT Instruction

For long operands:



For short operands:



The arctangent of the second operand is placed in the first-operand location. The operand and the result are floating-point numbers in the same format.

The result is in radians, normalized, and smaller in magnitude than *pi*/2. A nonzero result has the same sign as the operand. If the operand has a zero fraction, the result is a true zero.

If the normalized result would have a characteristic that is less than zero (exponent underflow), condition code 1 is set; the setting of the exponent-underflow mask bit in the PSW has no effect. If the second operand is outside the operand range implemented by the model, condition code 3 is set. In both cases where a nonzero condition code is set, the first-operand location remains unchanged.

If the operation is completed normally, condition code 0 is set.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:

0	Valid operation
1	Exponent underflow
2	--
3	Invalid operation

Program Exceptions:

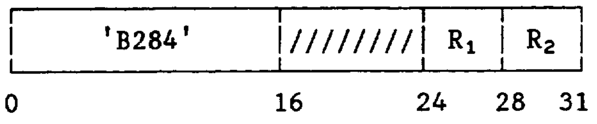
- Operation (if the arctangent facility is not installed)
- Specification

Programming Note

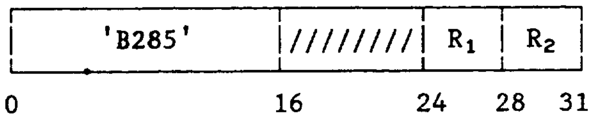
Exponent underflow can occur only when a nonzero second operand is unnormalized and so small that the result after normalization would have a characteristic less than zero.

COMMON LOGARITHM Instruction

For long operands:



For short operands:



The common logarithm (the logarithm to the base 10) of the second operand placed in the first-operand location. The operand and the result are floating-point numbers in the same format.

The result is normalized. If the operand is plus one, the result is a true zero.

If the second operand has a zero fraction or is negative, or if the second operand is outside the operand range implemented by the model, condition code 3 is set, and the first-operand location remains unchanged. Otherwise, the operation is completed normally, and condition code 0 is set.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:

0	Valid operation
1	--
2	--
3	Invalid operation

Program Exceptions:

- Operation (if the common-logarithm facility is not installed)
- Specification

Programming Note

Exponent underflow cannot occur, because the result is either zero or sufficiently greater than zero in magnitude to be representable as a normalized floating-point number.

COSINE Instruction

For long operands:

'B28A'	/////////	R_1	R_2
0	16	24	28 31

For short operands:

'B28B'	/////////	R_1	R_2
0	16	24	28 31

The cosine of the second operand in radians is placed in the first-operand location. The operand and the result are floating-point numbers in the same format.

The result is normalized and never greater in magnitude than one. If the operand has a zero fraction, the result is exactly plus one.

If the second operand is outside the operand range implemented by the model, or if the absolute value of the second operand is not less than *pi* multiplied by 2^{50} (long) or *pi* multiplied by 2^{18} (short), then the first-operand location remains unchanged, and condition code 3 is set. Otherwise, the operation is completed normally, and condition code 0 is set.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:

0	Valid operation
1	--
2	--
3	Invalid operation

Program Exceptions:

- Operation (if the sine-cosine facility is not installed)
- Specification

Programming Note

Mathematically, the cosine of *pi*/2, or of any odd multiple of *pi*/2, is zero. In practice, the floating-point operands nearest to those values are sufficiently different from a multiple of *pi*/2 that the result cannot be zero or produce exponent underflow.

EXPONENTIAL Instruction

For long operands:

'B282'	/////////	R_1	R_2
0	16	24	28 31

For short operands:

'B283'	/////////	R_1	R_2
0	16	24	28 31

The result of raising the mathematical constant *e* to the power of the second operand is placed in the first-operand location. The operand and the result are floating-point numbers in the same format.

The result is normalized. If the operand has a zero fraction, the result is exactly plus one.

If the normalized result would have a characteristic that is less than zero (exponent underflow), condition code 1 is set; the setting of the exponent-underflow mask bit in the PSW has no effect. If the normalized result would have a characteristic that is greater than 127 (exponent overflow), condition code 2 is set. If the second operand is outside the operand range implemented by the model, condition code 3 is set. In all three cases where a nonzero condition code is set, the first-operand location remains unchanged.

If the operation is completed normally, condition code 0 is set.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:

0	Valid operation
1	Exponent underflow
2	Exponent overflow
3	Invalid operation

Program Exceptions:

- Operation (if the exponential facility is not installed)
- Specification

NATURAL LOGARITHM Instruction

For long operands:

'B280'	/////////	R_1	R_2
0	16	24	28 31

For short operands:

'B281'	/////////	R_1	R_2
0	16	24	28 31

The natural logarithm (the logarithm to the base *e*) of the second operand is placed in the first-operand location. The operand and the result are floating-point numbers in the same format.

The result is normalized. If the operand is plus one, the result is a true zero.

If the second operand has a zero fraction or is negative, or if the second operand is outside the operand range implemented by the model, the first-operand location remains unchanged, and condition code 3 is set. Otherwise, the operation is completed normally, and condition code 0 is set.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:

0	Valid operation
1	--
2	--
3	Invalid operation

Program Exceptions:

- Operation (if the natural-logarithm facility is not installed)
- Specification

Programming Note

Exponent underflow cannot occur, because the result is either zero or sufficiently greater than zero in magnitude to be representable as a normalized floating-point number.

RAISE TO POWER Instruction

For long operands:

'B286'	/////////	R_1	R_2
0	16	24	28 31

For short operands:

'B287'	/////////	R_1	R_2
0	16	24	28 31

The first operand is raised to the power of the second operand, and the result is placed in the first-operand location. The operands and the result are floating-point numbers in the same format.

The result is normalized. If the first operand has a zero fraction and the second operand is greater than zero, the result is a true zero. If the first operand has a nonzero fraction and the second operand has a zero fraction, the result is exactly plus one.

For the operation to be valid, (1) the first operand must be greater than zero, or (2) if the first operand has a zero fraction, the second operand must be greater than zero, or (3) if the first operand is less than zero, the second operand must have a zero fraction.

If the normalized result would have a characteristic that is less than zero (exponent underflow), condition code 1 is set; the setting of the exponent-underflow mask bit in the PSW has no effect. If the normalized result would have a characteristic that is greater than 127 (exponent overflow), condition code 2 is set. If the operation is invalid, or if either operand is outside the operand range implemented by the model, condition code 3 is set. In all three cases where a nonzero condition code is set, the first-operand location remains unchanged.

If the operation is completed normally, condition code 0 is set.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:

0	Valid operation
1	Exponent underflow
2	Exponent overflow
3	Invalid operation

Program Exceptions:

- Operation (if the raise-to-power facility is not installed)
- Specification

Programming Note

Whether the operation is valid depends on the signs and on the zero or nonzero values of the two operands. Figure 5 summarizes the conditions under which the operation is invalid, causing condition code 3 to be set, or valid, causing condition code 0 to be set except when exponent underflow (code 1) or exponent overflow (code 2) occurs.

First Operand	Second Operand		
	< 0	= 0	> 0
< 0	I	N	I
= 0	I	I	Z
> 0	V	N	V

Explanation:

I	Invalid
N	Valid, result is one.
V	Valid
Z	Valid, result is zero.

Figure 5. Operand Validity

Not shown above is the possibility that condition code 3 is set because an otherwise valid operand is outside the range implemented by the model.

SINE Instruction

For long operands:

'B288'	////////	R_1	R_2
0	16	24	28 31

For short operands:

'B289'	////////	R_1	R_2
0	16	24	28 31

The sine of the second operand in radians is placed in the first-operand location. The operand and the result are floating-point numbers in the same format.

The result is normalized and never greater in magnitude than one. If the operand has a zero fraction, the result is a true zero.

If the normalized result would have a characteristic that is less than zero (exponent underflow), condition code 1 is set; the setting of the exponent-underflow mask bit in the PSW has no effect. If the second operand is outside the operand range implemented by the model, or if the absolute value of the second operand is not less than *pi* multiplied by 2^{50} (long) or *pi* multiplied by 2^{18} (short), then condition code 3 is set. In all those cases where a nonzero condition code is set, the first-operand location remains unchanged.

If the operation is completed normally, condition code 0 is set.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Resulting Condition Code:

0	Valid operation
1	Exponent underflow
2	--
3	Invalid operation

Program Exceptions:

- Operation (if the sine-cosine facility is not installed)
- Specification

Programming Note

Mathematically, the sine of *pi*, or of any multiple of *pi*, is zero. In practice, except for a zero operand, the floating-point operands nearest to those values are sufficiently different from a multiple of *pi* that the result cannot be zero or produce exponent underflow.

Exponent underflow can occur only when a nonzero second operand is unnormalized and so small that the result after normalization would have a characteristic less than zero.

INSTRUCTION CHARACTERISTICS

Figure 6 summarizes various characteristics of the mathematical-function instructions, including any restrictions on the operand and result range of each instruction, whether special conditions cause nonzero condition codes to be set, and the results produced by some instructions for certain unique operands. All instructions set condition code 0 when execution is completed normally, and all set condition code 3 when an operand is outside the operand range implemented by the model.

	Range Restrictions on		Condition Code			
Function	Operands	Result	1	2	3	Special Values
ARCTANGENT		$\lvert r \rvert \le pi/2$	EU^1			If x=0, then r=0
COMMON LOGARITHM	$x > 0$				$x \le 0$	If x=1, then r=0
COSINE	$\lvert x \rvert < L$	$\lvert r \rvert \le 1$				If x=0, then r=1
EXPONENTIAL		$r > 0$	EU	EO		If x=0, then r=1
NATURAL LOGARITHM	$x > 0$				$x \le 0$	If x=1, then r=0
RAISE TO POWER	(y>0)& any x	$r \ge 0$	EU	EO	$(y=0)\&(x\le0)$	If(y=0)&(x>0), r=0
(y**x)	(y=0)&(x>0)				$(y<0)\&(x\ne0)$	If$(y\ne0)$&(x=0), r=1
	(y<0)&(x=0)					
SINE	$\lvert x \rvert < L$	$\lvert r \rvert \le 1$	EU^1			If x=0, then r=0

Explanation:

&	"and"
EO	Exponent overflow
EU	Exponent underflow
EU^1	Exponent underflow can be caused only by a very small, unnormalized operand.
L	Limit: $pi*2^{50}$ for long operands, $pi*2^{18}$ for short operands
r	Result; placed in first-operand location
r=0	Result is true zero.
x	Operand; obtained from second-operand location
x=0	Second operand has zero fraction.
y	Operand (only for some instructions); obtained from first-operand location
y=0	First operand has zero fraction.

Figure 6. Instruction Characteristics

Name	Characteristics				Op Code
ARCTANGENT (long)	RRE C ZT	SP			B294
ARCTANGENT (short)	RRE C ZT	SP			B295
COMMON LOGARITHM (long)	RRE C LC	SP			B284
COMMON LOGARITHM (short)	RRE C LC	SP			B285
COSINE (long)	RRE C SN	SP			B28A
COSINE (short)	RRE C SN	SP			B28B
EXPONENTIAL (long)	RRE C EP	SP			B282
EXPONENTIAL (short)	RRE C EP	SP			B283
MULTIPLY AND ADD	RRE C MA	A SP	II	R ST	B243
NATURAL LOGARITHM (long)	RRE C LN	SP			B280
NATURAL LOGARITHM (short)	RRE C LN	SP			B281
RAISE TO POWER (long)	RRE C RP	SP			B286
RAISE TO POWER (short)	RRE C RP	SP			B287
SINE (long)	RRE C SN	SP			B288
SINE (short)	RRE C SN	SP			B289
SQUARE ROOT (long)	RRE QR	SP	SQ		B244
SQUARE ROOT (short)	RRE QR	SP	SQ		B245

Explanation:

A	Access exceptions for logical addresses
C	Condition code is set.
EP	Exponential facility
II	Interruptible instruction
LC	Common-logarithm facility
LN	Natural-logarithm facility
MA	Multiply-and-add facility
QR	Square-root facility
R	PER general-register-alteration event
RP	Raise-to-power facility
RRE	RRE instruction format
SN	Sine-cosine facility
SP	Specification exception
SQ	Square-root exception
ST	PER storage-alteration event
ZT	Arctangent facility

Figure 7. Summary of Mathematical-Assist Instructions

This page is intentionally left blank.

IBM System/370 Mathematical Assists
Order No. SA22-7094-1

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

____

What is your occupation? ______________________________

Number of latest Newsletter associated with this publication: ______________________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the front cover or title page.)

Note: Staples can cause prc) with automated mail sorting equipment.
Please use pressure-sensitive or other gummed tape to seal this form.

SA22-7094-1

Reader's Comment Form

Cut or Fold Along Line

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department B98
P.O. Box 390
Poughkeepsie, New York 12602

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

IBM System/370 Mathematical Assists (File No. S370-01) Printed in U.S.A. SA22-7094-1