GA22-7082-0
File No. S370-01

Systems

IBM Multiply-Add Facility

IBM

First Edition (January 1982)

Changes are periodically made 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.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Publications are not stocked at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Product Publications, Dept. B98, PO Box 390, Poughkeepsie, NY, U.S.A. 12602. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Preface

This publication describes the multiply-add facility that helps the processor perform specific mathematical computations, such as matrix multiplication, inversion, and decomposition. These computations occur in many applications, including finite element analysis, linear programming, and statistical analysis.

The facility consists of one instruction, MULTIPLY AND ADD, which may be used in library subroutines that perform such mathematical functions. The instruction may be used in either the System/370 mode or the ECPS:VSE mode, if the model provides the mode.

The reader should be familiar with the *IBM System/370 Principles of Operation,* GA22-7000, or the *IBM 4300 Processors Principles of Operation for ECPS:VSE Mode,* GA22-7070, as appropriate.

The facility discussed in this publication is not necessarily available on every model. The publication of this manual does not imply any intention by IBM to provide this facility on models other than those for which it is announced. For information concerning the availability of this facility on any specific model, refer to the latest edition of the functional characteristics manual for the specific model.

Contents

Multiply-Add Facility

Overview

The multiply-add facility consists of the MULTIPLY AND ADD instruction. This 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 x S) + C.
```

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. Scalar S is a floating-point number that previously was loaded into floating-point register 0. The floating-point numbers are all normalized and in the long format of 64 bits, except that vector C may contain unnormalized elements.

Vectors in Storage

The vectors in storage must be aligned on doubleword boundaries, so that their addresses are multiples of 8. The elements of a vector may be contiguous (in adjacent storage locations), or they may be spaced apart. 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 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 X N matrix that is stored in column order, the convention used for 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 types of vector 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 be the same. If either of the two operand vectors partially overlaps the result vector in storage, the effect is unpredictable.

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 vector elements:

```
LOAD (LD)              Load an element of B
MULTIPLY (MDR)         Multiply by scalar S
ADD NORMALIZED (AD)    Add an element of C
STORE (STD)            Store the 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 are processed.

The MULTIPLY AND ADD instruction differs from such a loop in that no interruptions occur when an arithmetic exception occurs, such as exponent overflow or underflow, even though the program mask in the PSW may permit the interruption. Instead, an arithmetic exception causes instruction execution to be completed and a nonzero condition code to be set before the result for the current elements is stored.

The MULTIPLY AND ADD instruction also differs in that execution is completed without storing the result element, and a nonzero condition code is set, when the instruction encounters an unnormalized multiplication operand or a vector element that is not correctly aligned in storage. If no such exceptional conditions are recognized, condition code 0 is set to indicate normal execution.

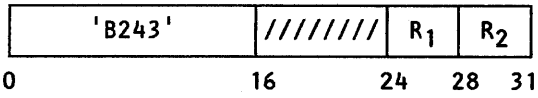
The MULTIPLY AND ADD instruction should be followed by a BRANCH ON CONDITION instruction to test for condition code 0. A nonzero condition code causes branching to a program of floating-point *scalar* instructions which use the same storage addresses in order to reprocess the current

vector elements. If the scalar instructions encounter the same exception, an interruption occurs, causing the same action that occurs when any other floating-point instruction is executed. 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 have other types of interruptions, such as page faults and I/O or external interruptions. As with the COMPARE LOGICAL LONG (CLCL) and MOVE LONG (MVCL) instructions, the instruction, when re-executed, resumes at the point of interruption.

MULTIPLY AND ADD Instruction

MULTIPLY AND ADD



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

$A = (B \times S) + C$

where A, B, and C are three vector operands and S is a scalar. 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 normalized numbers, but the elements of operand C may be normalized or unnormalized. 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 are fixed, 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. Bits 8-31 of 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). Bits 8-31 of the even R_1 register contain the element separation for operands A and C, and bits 8-31 of the odd R_1 register contain 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. The element separation is treated as a signed binary integer. 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, and instruction execution is suppressed.

The R_2 field, if nonzero, designates a general register, called the R_2 register; bits 8-31 of the R_2 register contain 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.

For all of these general registers, except general register 1, the contents of bits 0-7 are ignored.

Graphically, Figure 1 shows the contents of the general registers, where GR1, GR2, and GR3 represent general registers 1, 2, and 3, respectively.

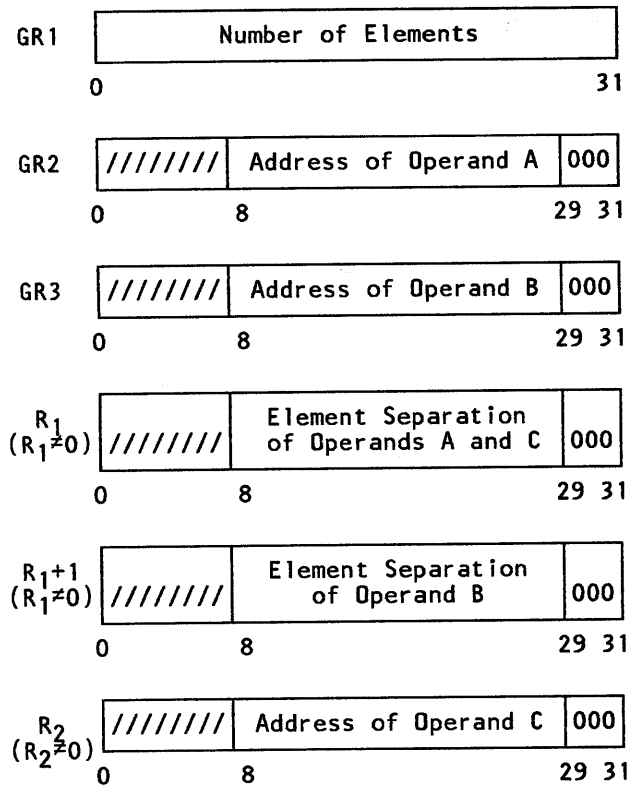


Figure 1. General-Register Assignment

Figure 2 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 2. Operand Addresses and Element Separations

Execution of the instruction begins with three tests performed in the order stated. 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 has a zero leftmost hexadecimal digit, 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 of the preceding tests results in a condition code being set, 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:

1. If the fraction of the element at the address of operand B is nonzero but has a zero as the leftmost hexadecimal digit, condition code 2 is set, and instruction execution is completed. If condition code 2 is not set, go to step 2.

2. If condition code 2 is not set, 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 exception is recognized during either the multiplication or the addition, it is not treated as a program-interruption condition. Instead, condition code 1 is set, and instruction execution is completed.

   If an exponent-underflow exception is recognized during either the multiplication or the addition, or if a significance exception is recognized during the addition, the exception is not treated as a program-interruption condition. 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 exception 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 second-operand 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, a significance exception causes a true zero to replace the result of the addition, and instruction execution continues.

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

Exponent overflow or underflow is not recognized during the multiplication 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 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 added to the contents of the R_2 register. For all these additions, carries out of bit position 8 are ignored, and bit positions 0-7 of the updated general registers are set to zeros. Floating-point register 0, the R_1 registers, and the elements in storage for operands B and C remain unchanged.

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.

For each set of elements, 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.

Execution of the instruction is interruptible for 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.

When general register 1 contains zero at the start of instruction execution, no access exceptions are recognized for any operand, the change bits for operand A remain unchanged, and no PER event for 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 result is unpredictable. The result is also unpredictable if $R_1 = 2$ or, for $R_2 \neq 0$, if:

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

Resulting Condition Code:

0	All elements are processed
1	Exponent overflow, exponent underflow, or significance loss
2	Unnormalized scalar or operand-B element
3	No elements are processed for other reasons

Program Exceptions:

Access (fetch, operands B and C; store, operand A)

Operation (if the facility is not installed)

Specification

Programming Notes

1. Unlike the scalar floating-point instructions MULTIPLY and ADD NORMALIZED, the MULTIPLY AND ADD instruction does not process operands that are not aligned on doubleword boundaries in storage. Moreover, 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. Instead, the instruction sets a condition code other than 0, so that scalar floating-point instructions may be used to perform the arithmetic.

   If execution of the MULTIPLY AND ADD instruction sets a condition code other than 0, 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 have been updated to address the next set of elements to be processed. This allows the program to issue scalar floating-point instructions that attempt to process these elements.

   If, during the execution of these scalar instructions, an exponent-overflow, exponent-underflow, or significance exception is recognized, a program interruption can occur which may invoke the same fixup routines that are used for all floating-point instructions. If an unnormalized operand is encountered or a vector in storage is unaligned, the scalar instructions can process the elements.

   Programming Note 2 contains a programming example.

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.

   The example uses the DC assembler instruction to define the appropriate MULTIPLY AND ADD instruction as a hexadecimal constant.

```
MADD EQU *
     DC  X'B2430046' Multiply and Add inst
     BC  8,CONT      Test condition code 0
* Do following instructions if CC not 0
     LD  2,0(,3)     Load element of B
     MDR 2,0         Multiply by S
     AD  2,0(,6)     Add element of C
     STD 2,0(,2)     Store element of A
     LA  2,0(2,4)    Update address of A
     LA  3,0(3,5)    Update address of B
     LA  6,0(6,4)    Update address of C
     BCT 1,MADD      Branch if not done
CONT 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 the BRANCH ON CONDITION instruction and by a loop containing the 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 the preceding MULTIPLY AND ADD instruction, except as follows:

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

b. The multiplication operands may be unnormalized numbers.

c. Exponent overflow causes an interruption; and exponent underflow and significance loss cause an interruption, if permitted by the PSW masks.

d. A floating-point working register is needed.

e. The preceding example shows no initial test for a zero number of elements in general register 1 because the loop is not entered.

For the special instances in which C equals A or in which the vector elements are contiguous, the program may be simplified accordingly.

3. Any two or all three vectors may be the same, if both the addresses and the 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. The machine does not check for partial overlap with the result vector.

   If vector C is the same as 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, register 3 may be updated either once or twice for each element, depending on the model. Similarly, the R_2 field should not designate general register 2 if C equals A. Thus, an R_2 field containing 2 may or may not have the same effect as an R_2 field containing 0.

4. The instruction should not be used to store data into its own location. The effect is unpredictable because the instruction may be refetched from storage and reinterpreted, even in the absence of an interruption during execution. The exact point in the execution at which such a refetch may occur is unpredictable.

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.

IBM Multiply-Add Facility
Order No. GA22-7082-0

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:

_____

_____

_____

_____

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

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.)

Reader's Comment Form

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:

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

IBM®

Cut or Fold Along Line

IBM Multiply-Add Facility (File No. S370-01) Printed in U.S.A. GA22-7082-0

Cut or Fold Along Line

IBM Multiply-Add Facility (File No. S370-01) Printed in U.S.A. GA22-7082-0

GA22-7082-0