HONEYWELL EDP

HARDWARE BULLETIN

SERIES 200

SCIENTIFIC UNIT FOR MODELS 1200 AND 2200 (FEATURE 1100)

SUBJECT: Data Format and Programming Procedures for the Scientific Instructions Provided by Feature 1100.

SPECIAL INSTRUCTIONS: This hardware bulletin augments the Honeywell Series 200 Programmers' Reference Manual, Models 200/1200/2200 for users of Model 1200 or 2200 computers equipped with the Scientific Unit (Feature 1100). The reader is assumed to be familiar with the contents of the reference manual, which has the file control number 113.0005.0000.00.00. For added convenience, the information presented herein is summarized in Appendix F of the manual.

DATE: September 30, 1965

FILE NO.: 112.0005.1539.00.00

8454
5965
Printed in U.S.A.

Questions and comments regarding this manual should be addressed to:

Honeywell Electronic Data Processing
Information Services
60 Walnut Street
Wellesley Hills, Massachusetts 02181

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION I
INTRODUCTION

The scientific unit (Feature 1100) may be attached to the Type 1201 or 2201 processor. The following types of scientific instructions are provided:

1. Floating-point load and store.
2. Floating-point arithmetic.
3. Decimal-to-binary and binary-to-decimal conversion.
4. Floating-point test and branch.
5. Binary integer arithmetic.
6. Mantissa shift.

FLOATING-POINT DATA FORMAT

A floating-point number is represented by a fixed-length, 48-bit word. The high-order 36 bits contain a fraction, the mantissa. The low-order 12 bits contain an exponent of base 2. The value of a floating-point number is the product of the mantissa and 2 raised to the indicated exponent. As explained below, a Series 200 floating-point word is capable of expressing numbers in the range $\pm 2^{-2048}$ to $\pm 2^{+2047}$, or approximately $\pm 10^{\pm 616}$. In main memory, a floating-point word occupies a field of eight consecutive character positions, as shown in Figure 1-1.

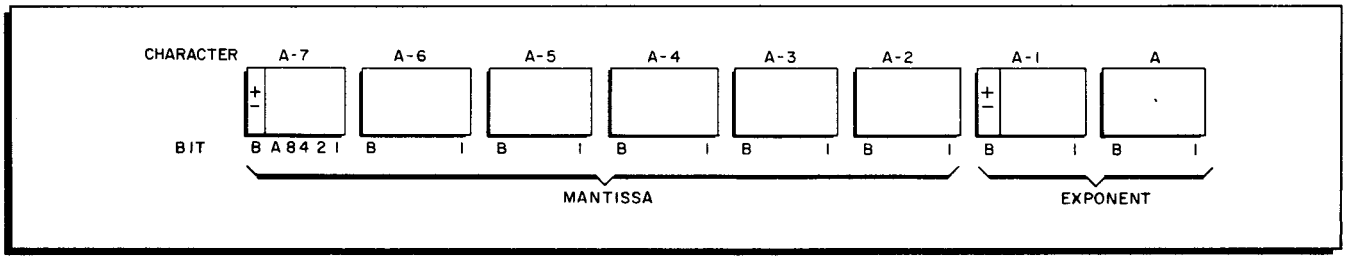


Figure 1-1. Main Memory Floating-point Data Format

Four floating-point accumulators are reserved in control memory to contain operands and results of floating-point operations. The accumulators are explicitly addressed in the floating-point instructions by the octal digits 0, 1, 2, and 3. Each accumulator is composed of three specific, 18-bit, control memory registers, as explained below. Only the low-order 12 bits of the rightmost register are used to express the exponent. Figure 1-2 illustrates the floating-point accumulator data format.

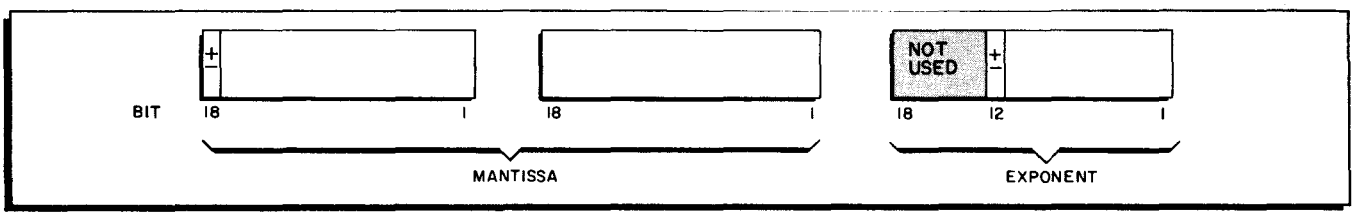


Figure 1-2. Floating-point Accumulator Data Format

FLOATING-POINT NUMERICAL REPRESENTATION

The Series 200 floating-point word is expressed in twos-complement binary notation. That is, the mantissa is a binary fraction, the exponent is a binary integer, and negative mantissas and exponents are expressed as the twos complements of the positive values.

The twos complement of a binary number is formed by:

1. Subtracting each bit position from 1 (equivalent to changing all 1's to 0's and vice versa); then
2. Adding 1 to the low-order (units) bit position.

For example, to find the twos complement of 011, change 1's to 0's and 0's to 1's, giving 100. Then add a binary 1 to give 101. Now to determine the original number, simply recomplement the twos-complement number formed above.

$$\begin{array}{l} 101 \longrightarrow 010 \\ \qquad\quad \underline{\quad 1} \\ \qquad\quad 011 \end{array}$$

Using twos-complement notation to represent negative numbers facilitates floating-point arithmetic operations. In a subtraction operation, the twos complement of the subtrahend is added to the minuend. Since multiplication and division are actually successive addition or subtraction operations, all twos-complement arithmetic is accomplished by one or more additions.

Table 1-1 below specifies the numerical representation of mantissas. In twos-complement notation, only the low-order 35 bits are used to represent positive mantissas; the high-order bit is always zero. Negative mantissa values are expressed as the twos complement of the corresponding positive values, always forcing the high-order bit to 1. Consequently, the high-order bit in twos-complement notation is a sign bit — 0 for positive and 1 for negative. As mentioned above, the absolute value of a negative number is found by recomplementation. Note that the mantissa is a fraction. There is an implied binary point to the right of the sign bit.

Numerical representation of exponents is shown in Table 1-2. A positive exponent is a 12-bit binary integer whose high-order bit is 0. A negative exponent is a 12-bit binary twos-complement integer whose high-order bit, by definition, is 1.

Table 1-1. Floating-point Numerical Representation of Mantissas

Sign → ; implied binary point				Mantissa Value
Bit Position:	36	35	34----------2 1	
Bit Value:	2^0 .	2^{-1}	2^{-2} ... 2^{-34}	2^{-35}
	0	1	1----------1	1	$+1-2^{-35}$
			. . .		
	0	1	0----------0	0	+1/2
	0	0	1----------1	1	$+1/2-2^{-35}$
			. . .		
	0	0	0----------0	1	$+2^{-35}$
	0	0	0----------0	0	+0
	1	1	1----------1	1	-2^{-35}
			. . .		
	1	1	0----------0	0	-1/2
	1	0	1----------1	1	$-(1/2+2^{-35})$
			. . .		
	1	0	0----------0	0	-1

Table 1-2. Floating-point Numerical Representation of Exponents

Sign →				Exponent Value
Bit Position	12	11	10----------2 1	
Bit Value:	2^{11}	2^{10}	2^9 ... 2^1	2^0
	0	1	1----------1	1	±2047
			. . .		
	0	0	0----------0	1	+1
	0	0	0----------0	0	+0
	1	1	1----------1	1	-1
	1	1	1----------1	0	-2
			. . .		
	1	0	0----------0	0	-2048

Floating-point arithmetic instructions deliver results with normalized mantissas. For positive numbers, a normalized mantissa has a 1 immediately following the implied binary point (i.e., the high-order two bits are 01). For negative numbers, a normalized mantissa has a 0 immediately following the implied binary point (i.e., the high-order two bits are 10). In Table 1-1, normalized mantissas are shaded. A normal zero is defined as a floating-point word whose mantissa and exponent are both +0.

FLOATING-POINT REGISTERS

The four addressable floating-point accumulators occupy the following locations in control memory:

Accumulator Address	Control Memory Location (Operator's Control Panel Only)		
	High-Order Mantissa	Low-Order Mantissa	Exponent
0	43	42	41
1	47	46	45
2	53	52	51
3	57	56	55

NOTE: In program instructions, the floating-point accumulators may be addressed only via the octal digits 0, 1, 2, and 3 in the floating-point instructions. The instructions LCR and SCR must not be used to address these accumulators. At the control panel, the operator may address these locations with the addresses in the above table.

A "pseudo accumulator" is provided, which always contains a normal zero. The pseudo accumulator is addressed by the octal digit 7. Any floating-point number may be normalized by adding it to the normal zero in accumulator 7. Note that the pseudo accumulator should not be specified as the result location in any floating-point instruction, because the result data will be lost.

The scientific unit also includes a low-order result register (LOR). The LOR may contain a low-order sum, difference, or product, or the remainder of a division operation. In effect, the LOR provides an additional 36 bits of mantissa precision. The LOR is not addressed explicitly in the floating-point arithmetic instructions, as are the accumulators. However, instructions are provided to load and store the contents of the LOR.

SCIENTIFIC UNIT INDICATORS

Three indicators are present in the scientific unit. The exponent overflow indicator is activated when a base-2 exponent exceeds +2047. The actual result delivered to the result

accumulator when the exponent overflow condition is present contains a correct mantissa and an exponent which is 4096 less than the correct exponent.

NOTE: When an exponent becomes less than -2048, a normal zero is delivered and no indication is given.

The divide check indicator is activated when a divisor is equal to zero. When the divide check condition is present, the division operation is not executed. The multiply overflow indicator is activated when the product of a Binary Integer Multiply instruction exceeds 24 bits in length. When the multiply overflow condition is present, the low-order 24 bits are delivered as the result, and the high-order bits are lost. The above indicators may be tested by the Floating Test and Branch on Indicator instruction described in Section V.

AUTOMATIC FORMATTING IN ARITHMETIC OPERATIONS

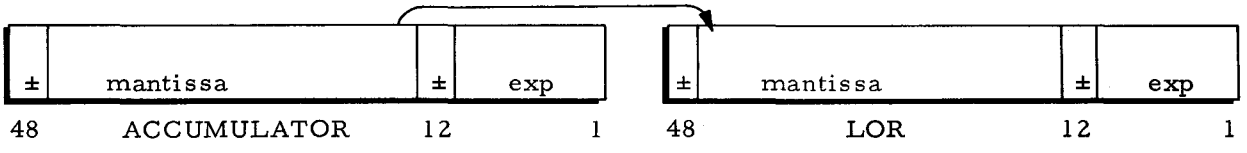
Floating-point arithmetic instructions accept either normalized or unnormalized operands. The scientific unit automatically shifts operands in order to perform arithmetic operations, and automatically normalizes results of arithmetic operations. The three types of automatic formatting are described below.

Prenormalization

In a floating divide operation, an unnormalized divisor is prenormalized. The mantissa is left-shifted until normalized, and the exponent is decreased by one for each bit position shifted.

Equalization

In floating add and subtract operations, equalization occurs after prenormalization. The mantissa of the operand with the smaller exponent is right-shifted, and the exponent is increased by one for each bit position shifted, until the exponents of the two operands are equal. Bits are shifted from the low-order mantissa position of the accumulator (bit 13) into the high-order mantissa position of the LOR (bit 47), as shown below.



Postnormalization

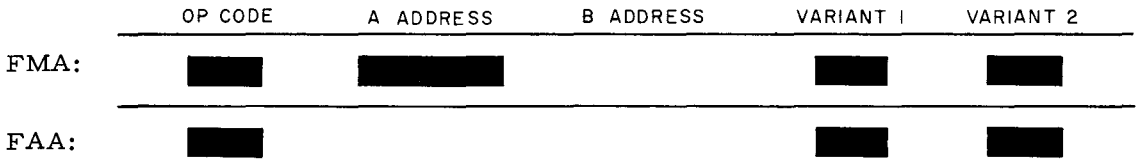
The results of floating add, subtract, multiply, and divide operations are normalized. If the tentative result is unnormalized, the mantissa is left-shifted until normalized, and the exponent is decreased by one for each bit position shifted. For results in which mantissa overflow occurred, the mantissa is right-shifted one bit position and the exponent is increased by one. Note that postnormalization may restore bits which were shifted into the LOR by equalization.

INSTRUCTION FORMATS

Only four operation codes are associated with the 14 scientific instructions. The Binary Mantissa Shift instruction has the mnemonic BMS (octal code 04). The Binary Integer Multiply instruction has the mnemonic BIM (octal code 05). All the remaining floating-point instructions use one or both of the following op codes:

Name	Mnemonic	Octal Code
Floating Memory to Accumulator	FMA	07
Floating Accumulator to Accumulator	FAA	06

The full formats of the floating-point instructions are given below:



The first six-bit instruction variant usually addresses the floating-point accumulators used in an operation. In subsequent instruction descriptions, this variant is abbreviated

X	Y

,

where octal digits X and Y are the accumulator addresses given on page 1-4. The accumulator X addressed in the high-order three variant bits is usually the source of a floating-point operand. The accumulator Y addressed in the low-order three variant bits is usually the destination of a floating-point result. The second instruction variant is a six-bit octal character which defines the particular floating-point instruction (e.g., Floating Multiply).

The memory-to-accumulator format is used in those instructions which require a main memory address in addition to floating-point accumulator references. In instruction descriptions, the A address of an instruction is abbreviated by the letter A. The A address may define the main memory location of an 8-character, floating-point operand, or it may specify a branch address. The accumulator-to-accumulator format is used in those instructions which require only floating-point accumulator references.

In addition to the full instruction formats described above, each form of a floating-point instruction using the FMA or FAA format is assigned a unique assembly language mnemonic, which also generates the 06 or 07 octal op code. When an instruction is coded using its unique mnemonic, the second variant is automatically generated and is not written in the operands field by the programmer. In summary, the floating-point instructions may be coded in two equivalent forms:

1. The full form which contains an FMA or FAA mnemonic op code, an A address if appropriate, and two variants.

2. The unique form, which contains a unique mnemonic op code, an A address if appropriate, and one variant.

Both forms are described for each instruction in the following sections.

PROGRAMMING CONSIDERATIONS

For instructions in the FMA format, the A address is processed by the central processor in the usual manner, using the A-address register (AAR). The description of each instruction gives the address register settings after the operation. During instruction extraction, the two variants of FMA and FAA instructions are transmitted directly to the scientific unit. The variant register in the central processor is unaffected by these instructions. In the extraction or restoration of operands in memory, the scientific unit neither recognizes nor alters punctuation bits.

SYMBOLOGY

A:	A address of the instruction.
B:	B address of the instruction.
X:	Floating-point accumulator addressed in the high-order three bits of an instruction variant (usually the source of an operand).
Y:	Floating-point accumulator addressed in the low-order three bits of an instruction variant (usually the destination of a result).
X-:	In the first variant of an instruction, only the high-order three bits specifying accumulator X are significant.
-Y:	In the first variant of an instruction, only the low-order three bits specifying accumulator Y are significant.
(X) or (Y):	Floating-point word contained in accumulator X or Y.
LOR:	Low-order result register.
(LOR):	Floating-point word contained in LOR.
AAR:	A-address register.
BAR:	B-address register.
SR:	Sequence register.
A_p:	Previous setting of A-address register.
B_p:	Previous setting of B-address register.
JI:	Address of next instruction if branch occurs.
NXT:	Next sequential instruction.
N_n:	Number of automatic formatting shifts in an operation.
N_1:	Number of binary ones in a multiplier.

N_s: Number of shifts.

[ ] "smallest integer greater than"

N_i: Number of characters in an instruction.

TIMING NOTES

All timings shown are for Model 2200 and are based on the use of direct addressing. Three memory cycles should be added for each indexed address and one memory cycle should be added for each character extracted as a result of indirect addressing.

SECTION II

DATA MOVING INSTRUCTIONS

STORE FLOATING ACCUMULATOR

FORMAT

FMA/A, X-, 00 or TAM/A, X-

FUNCTION

(X) is stored in memory locations A through A-7.
(X) is unaltered.

TIMING[1]

N_i + 10 cycles.

REGISTERS AFTER OPERATION

AAR	BAR
A-8	B_p

EXAMPLE

Store the contents of floating accumulator 1 in the main memory field whose rightmost character is tagged RESULT.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

	CARD NUMBER	TYPE	MAR	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					FMA	RESULT,1Ø,ØØ OR	
2					TAM	RESULT,1Ø	
3							
4							

LOAD FLOATING ACCUMULATOR

FORMAT

FMA: FMA/A, -Y, 02 or TMA/A, -Y

FAA: FAA/XY, 02 or TAA/XY

[1] This and subsequent timings pertain to Model 2200.

FUNCTION

FMA: The floating-point word in memory locations A through A-7 is loaded into accumulator Y.

FAA: (X) is loaded into accumulator Y.

TIMING

FMA: N_i + 11 cycles

FAA: 8 cycles

REGISTERS AFTER OPERATION

	AAR	BAR
FMA:	A-8	B_p
FAA:	A_p	B_p

NOTES

1. No normalization occurs.

EXAMPLES

1. Load the floating-point word stored in memory locations DELTA-7 through DELTA into floating accumulator 0.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62 63 ... 80
1				FMA	DELTA,ØØ,Ø2 OR
2				TMA	DELTA,ØØ

2. Load the contents of accumulator 3 into accumulator 0.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62 63 ... 80
1				FAA	3Ø,Ø2 OR
2				TAA	3Ø

STORE LOW-ORDER RESULT

FORMAT

FMA: FMA/A, 00, 07 or TLM/A

FAA: FAA/-Y, 07 or TLA/-Y

FUNCTION

FMA: (LOR) is stored in memory locations A through A-7.

FAA: (LOR) is stored in accumulator Y.

TIMING

FMA: N_i + 9 cycles

FAA: 6 cycles

REGISTERS AFTER OPERATION

	AAR	BAR
FMA:	A-8	B_p
FAA:	A_p	B_p

NOTE

1. No normalization occurs.

EXAMPLES

1. Store the contents of the LOR in the main memory field whose rightmost character is tagged RESULT.

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1					FMA	RESULT,ØØ,Ø7 OR
2					TLM	RESULT

2. Store the contents of the LOR in accumulator 2.

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1					FAA	Ø2,Ø7 OR
2					TLA	Ø2

LOAD LOW-ORDER RESULT

FORMAT

FMA: FMA/A, 00, 01 or TML/A

FAA: FAA/X-, 01 or TAL/X-

FUNCTION

FMA: The floating-point word in memory locations A through A-7 is loaded into the LOR.

FAA: (X) is loaded into the LOR.

TIMING

FMA: N_i + 9 cycles

FAA: 6 cycles

REGISTERS AFTER OPERATION

	AAR	BAR
FMA:	A-8	B_p
FAA	A_p	B_p

NOTE

1. No normalization occurs.

EXAMPLES

1. Load the floating-point word stored in memory locations STORE-7 through STORE into the LOR.

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
	1 2 3 4 5	6	7	8 14	15 20	21 62 63 80
1					FMA	STORE,ØØ,Ø1 OR
2					TML	STORE

2. Load the contents of accumulator 2 into the LOR.

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
	1 2 3 4 5	6	7	8 14	15 20	21 62 63 80
1					FAA	2Ø,Ø1 OR
2					TAL	2Ø

SECTION III

FLOATING-POINT ARITHMETIC INSTRUCTIONS

FLOATING ADD

FORMAT

FMA: FMA/A, XY, 10 or AMA/A, XY

FAA: FAA/XY, 10 or AAA/XY

FUNCTION

FMA: The floating-point word in memory locations A through A-7 is added to (X), and the sum is stored in accumulator Y. The low-order sum is stored in LOR.

FAA: (X) is added to (Y), and the sum is stored in accumulator Y. The low-order sum is stored in LOR.

TIMING

FMA: $N_i + 13 + \left[N_n/4\right]$ cycles

FAA: $11 + \left[N_n/4\right]$ cycles

REGISTERS AFTER OPERATION

	AAR	BAR	LOR
FMA:	A-8	B_P	The low-order result of the addition. The sign bit of LOR = 0. The exponent of LOR = the exponent of the high-order result minus 35.
FAA:	A_p	B_p	same as above

NOTES

1. Equalization, and postnormalization occur if required.
2. X and Y may specify the same accumulator.
3. An exponent overflow indication may be given.
4. A result with a zero mantissa is returned as a normal zero.

EXAMPLE

Add the three floating-point numbers stored in sequential fields beginning in location DATA. Store the sum in the eight-character field whose rightmost character is tagged SUM.

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1				FMA	DATA+7,Ø1,Ø2	load first no. into accumulator
2				FMA	DATA+15,11,1Ø	add second no.
3				FMA	DATA+23,11,1Ø	add third no.
4				FMA	SUM,1Ø,ØØ	store sum

FLOATING SUBTRACT

FORMAT

FMA: FMA/A, XY, 11 or SMA/A, XY

FAA: FAA/XY, 11 or SAA/XY

FUNCTION

FMA: The floating-point word in memory locations A through A-7 is subtracted from (X); i.e., its twos complement is added to (X). The result is stored in accumulator Y. The low-order result is stored in the LOR.

FAA: (Y) is subtracted from (X). The result is stored in accumulator Y, and the low-order result is stored in the LOR.

TIMING

FMA: $N_i + 13 + \left[N_n/4\right]$ cycles

FAA: $11 + \left[N_n/4\right]$ cycles

REGISTERS AFTER OPERATION

	AAR	BAR	LOR
FMA:	A-8	B_p	Low-order difference. Sign bit = 0. Exponent = high-order exponent minus 35.
FAA:	A_p	B_p	same as above.

NOTES

1. Equalization, and postnormalization occur if required.
2. X and Y may specify the same accumulator.
3. An exponent overflow indication may be given.
4. A result with a zero mantissa is returned as a normal zero.

EXAMPLE

1. Subtract the floating-point word in locations DATA-7 through DATA from the contents of accumulator 3 and store the result in accumulator 1.

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1				FMA	DATA,3 1 , 1 1 OR	
2				SMA	DATA,3 1	

FLOATING MULTIPLY

FORMAT

FMA: FMA/A, XY, 13 or MAM/A, XY

FAA: FAA/XY, 13 or MAA/XY

FUNCTION

FMA: (X) is multiplied by the floating-point word in memory locations A through A-7. The high-order product is stored in accumulator Y. The low-order product is stored in LOR.

FAA: (X) is multiplied by (Y). The high-order product is stored in accumulator Y. The low-order product is stored in LOR.

TIMING

FMA: $N_i + 21 + \left[N_1/2\right] + \left[N_n/4\right]$ cycles

FAA: $19 + \left[N_1/2\right] + \left[N_n/4\right]$ cycles

REGISTERS AFTER OPERATION

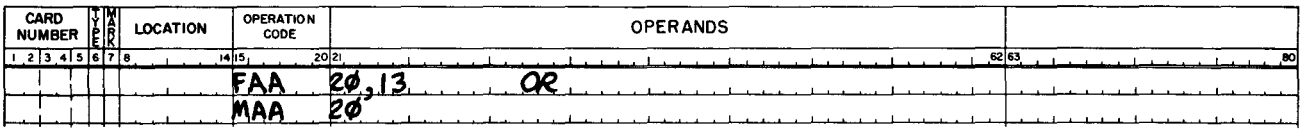
	AAR	BAR	LOR
FMA:	A-8	B_p	Low-order product. Sign bit = 0. Exponent = high-order exponent minus 35.
FAA:	A_p	B_p	Same as above.

NOTES

1. X and Y may specify the same accumulator.
2. Postnormalization occurs if required.
3. An exponent overflow indication may be given.
4. If either operand is equal to zero, the results in both accumulator and LOR are normal zeros.

EXAMPLE

1. Multiply the floating-point word in accumulator 2 by the floating-point word in accumulator 0, and store the product in accumulator 0.



FLOATING DIVIDE

FORMAT

FMA: FMA/A,XY,12 or DMA/A,XY

FAA: FAA/XY,12 or DAA/XY

FUNCTION

FMA: The floating-point word in locations A through A-7 is divided by (X). The quotient is stored in accumulator Y. The remainder is stored in LOR.

FAA: (Y) is divided by (X). The quotient is stored in accumulator Y. The remainder is stored in LOR.

TIMING

FMA: $N_i + 40 + \left[N_n/4\right]$ cycles

FAA: $38 + \left[N_n/4\right]$ cycles

REGISTERS AFTER OPERATION

	AAR	BAR	LOR
FMA:	A-8	B_p	Contains the remainder. The absolute value of the remainder mantissa is less than the absolute value of the mantissa of the normalized divisor. The sign of the remainder is equal to the sign of the dividend. The exponent of the remainder is equal to the exponent of the dividend minus 35, and plus one if the absolute value of the dividend mantissa is greater than the absolute value of the mantissa of the normalized divisor.
FAA:	A_p	B_p	same as above.

NOTES

1. Prenormalization of the divisor and postnormalization of the quotient occur if required.

2. X and Y may specify the same accumulator.

3. The quotient or remainder may cause an exponent overflow indication to be given.

4. If the divisor is zero, a divide check indication is given. The division is not executed, and accumulator Y is unaltered.

5. If the dividend is zero, the quotient and remainder are normal zeros.

EXAMPLES

1. Divide the floating-point word stored in the memory field whose rightmost character is tagged DATA by the floating-point word in accumulator 0. Store the quotient in accumulator 0.

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1				FMA	DATA,ØØ,12 OR	
2				DMA	DATA,ØØ	

2. Divide the floating-point word in accumulator 2 by the floating-point word in accumulator 3 and store the quotient in accumulator 2.

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1				FAA	32,12 OR	
2				DAA	32	

SECTION IV

DATA CONVERSION INSTRUCTIONS

DECIMAL TO BINARY CONVERSION

FORMAT

FMA/A, -Y, 03 or DTB/A, -Y

FUNCTION

The 11-character main memory field whose low-order character position is A is treated as a signed decimal integer. That is, each character represents a decimal digit. The sign of the integer is given by the zone bits of the units position (character A), as follows: 10 = negative; anything else = positive. The decimal integer is converted to a 36-bit binary integer and stored in the mantissa portion of (Y); the exponent of (Y) is set to +35.

TIMING

N_i + 24 cycles

REGISTERS AFTER OPERATION

AAR	BAR	LOR
A-11	B_p	Low-order result of conversion (see note 2 below). Sign bit = 0. Exponent = high-order exponent minus 35.

NOTES

1. The zone bits of the 10 high-order decimal characters are ignored. If the middle two data bits of any character are 11, that character is interpreted as a zero.

2. Because an 11-digit decimal number has a range of ± 99, 999, 999, 999 and a 36-bit binary twos-complement number has a range of approximately ± 34, 359, 738, 368, mantissa overflow of up to two bits is possible. If mantissa overflow occurs, the low-order one or two bits are shifted into LOR. Accumulator Y then contains the high-order result of conversion, with an exponent of 36 or 37. Note that when a low-order result is shifted into LOR, the high-order result is automatically normalized.

EXAMPLE

Convert 899,473 to a binary integer in the mantissa portion of accumulator 0.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1			DEC	DCW	+ØØØØØ899473
2				FMA	DEC,ØØ,Ø3

BINARY TO DECIMAL CONVERSION

FORMAT

FMA/A,X-,06 or BTD/A,X-

FUNCTION

The mantissa portion of (X) is converted from a twos-complement binary integer to a signed decimal integer. The decimal integer is stored in the 11-character main memory field whose low-order character is location A.

TIMING

N_i + 23 cycles

REGISTERS AFTER OPERATION

AAR	BAR
A-11	B_p

NOTES

1. If the binary integer is negative, the zone bits of the units character (location A) are set to 10. If the binary integer is positive, the zone bits of the units character are set to 01. The zone bits of the other 10 characters are set to 00.
2. The exponent in accumulator X is ignored and unaltered.

EXAMPLE

1. Convert the mantissa portion of the floating-point word in accumulator 3 to a signed decimal integer. Store the decimal integer in the main memory field whose rightmost character is tagged DEC.

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1				BTD	DEC,3Ø
2			DEC	DCW	#11CØØØØØØØØØØØØØØØØØØØØØ

SECTION V

CONTROL INSTRUCTIONS

FLOATING TEST AND BRANCH ON ACCUMULATOR CONDITION

FORMAT

FMA/A, XC, 04 or FBA/A, XC

FUNCTION

The mantissa portion of (X) is tested for the condition specified by C, the low-order octal digit of variant 1:

C = 0	no branch
C = 1	(X) = 0
C = 2	(X) < 0
C = 3	(X) ≤ 0
C = 4	(X) > 0
C = 5	(X) ≥ 0
C = 6	(X) = 0
C = 7	unconditional branch

If the condition specified by C is satisfied, program control branches to location A.

TIMING

N_i + 4 cycles	NO BRANCH
N_i + 6 cycles	BRANCH

REGISTERS AFTER OPERATION

AAR	BAR	SR	
A	B_p	NXT	NO BRANCH
A	NXT	JI(A)	BRANCH

NOTE

1. (X) must be normalized.

EXAMPLE

Subtract the floating-point word in accumulator 1 from the floating-point word in accumulator 0. If the difference is less than or equal to zero, branch to location LESS.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1					SAA	Ø1 floating subtract	
2					FBA	LESS,13 test and branch	

FLOATING TEST AND BRANCH ON INDICATOR

FORMAT

FMA/A,0D,05 or FBI/A,0D

FUNCTION

The indicator(s) specified by D, the low-order octal digit of variant 1, are tested. If any of the indicators is set, control branches to location A.

D = 0	no branch
D = 1	multiply overflow
D = 2	exponent overflow
D = 3	exponent and multiply overflow
D = 4	divide check
D = 5	divide check and multiply overflow
D = 6	divide check and exponent overflow
D = 7	divide check and exponent overflow, and multiply overflow

TIMING

N_i + 2 cycles	NO BRANCH
N_i + 4 cycles	BRANCH

REGISTERS AFTER OPERATION

AAR	BAR	SR	
A	B_p	NXT	NO BRANCH
A	NXT	JI(A)	BRANCH

NOTE

1. All indicators tested are reset.

EXAMPLE

Multiply the floating-point word in accumulator 1 by the floating-point word in accumulator 2. If exponent overflow occurs, store the contents of the sequence register and accumulator 2, replace the contents of accumulator 2 with the largest positive floating-point number, and continue.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1				FAA	12,13	floating multiply
2			TEST	FBI	OVER,Ø2	test for exponent overflow
3						
4						
5						
6			OVER	SCR	SEQREG,77	store sequence register
7				FMA	ACC,2Ø,ØØ	store accumulator
8				FMA	MAX,Ø2,Ø2	load accumulator with max. value
9				B	TEST+7	return (in four-char. mode)
10			SEQREG	DCW	#4CØØØØØØØØ	
11			ACC	DCW	#8CØØØØØØØØØØØØØØØØ	
12			MAX	DCW	#8C377777777777377777	

BINARY MANTISSA SHIFT

FORMAT

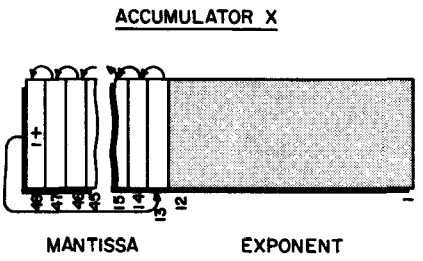
BMS/XM, V

FUNCTION

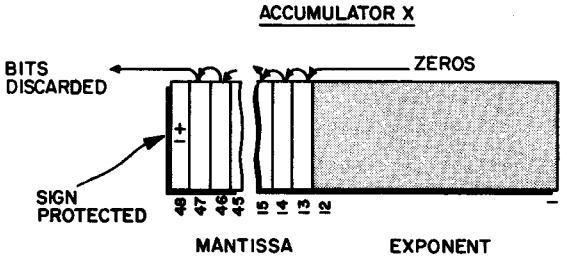
In a single-precision shift, the mantissa portion of (X) is shifted by the number of bit positions specified by variant 2 ($0 \leq V \leq 63$). In a double-precision shift, the mantissa portions of (X) and (LOR) are treated as a single register and shifted the number of bit positions specified by variant 2. The exponent portions of (X) and (LOR) are never shifted. A shift operation may be of either the rotate or the arithmetic type, in the left or right direction. In a rotate shift, bits shifted off the end of a "register" (mantissa of X or mantissas of X and LOR) are moved end-around to the opposite end of the register. That is, no bits are lost in a rotate shift. In an arithmetic shift, bits shifted off the end of a register are lost. Note that in an arithmetic shift, the sign positions of accumulator X and LOR are protected; i.e., bits are shifted around these positions. In a right arithmetic shift, the sign bit is duplicated in the vacated bit positions. In a left arithmetic shift, vacated bit positions are filled with zeros.

M, the low-order octal digit of variant 1, specifies the mode of shifting, as illustrated below.

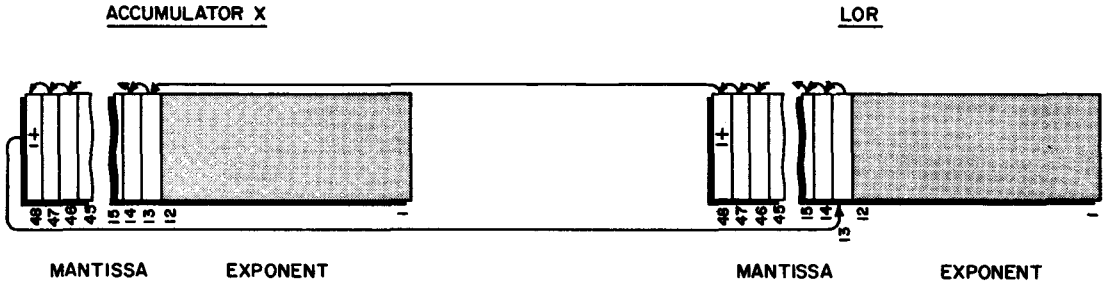
M = Ø : LEFT, ROTATE, SINGLE-PRECISION SHIFT



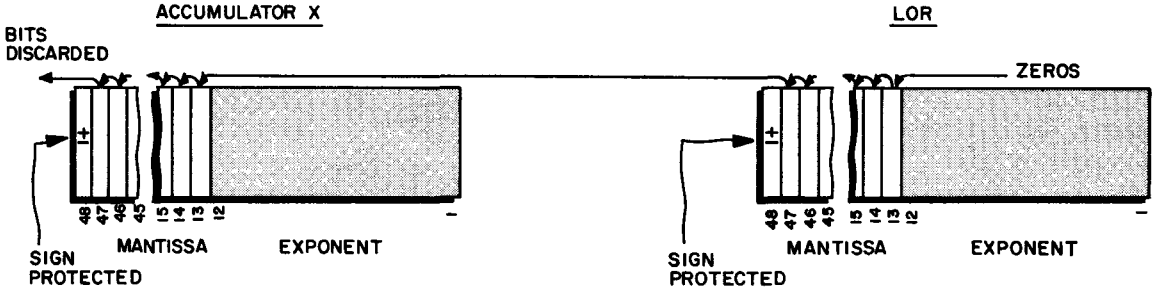
M = 1 : LEFT, ARITHMETIC, SINGLE-PRECISION SHIFT



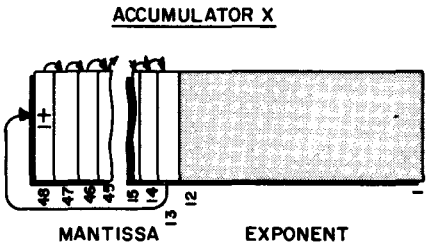
M = 2: LEFT, ROTATE, DOUBLE-PRECISION SHIFT



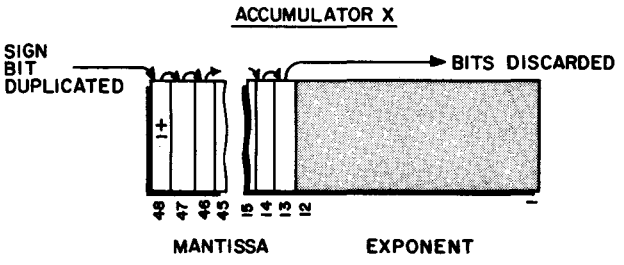
M = 3 : LEFT, ARITHMETIC, DOUBLE-PRECISION SHIFT



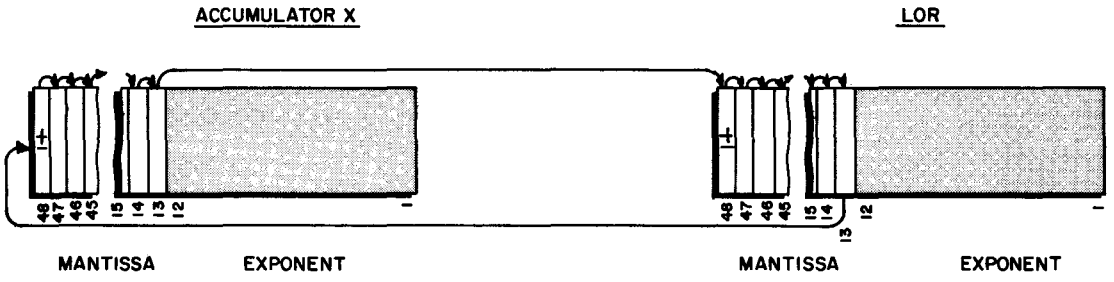
M=4: RIGHT, ROTATE, SINGLE-PRECISION SHIFT



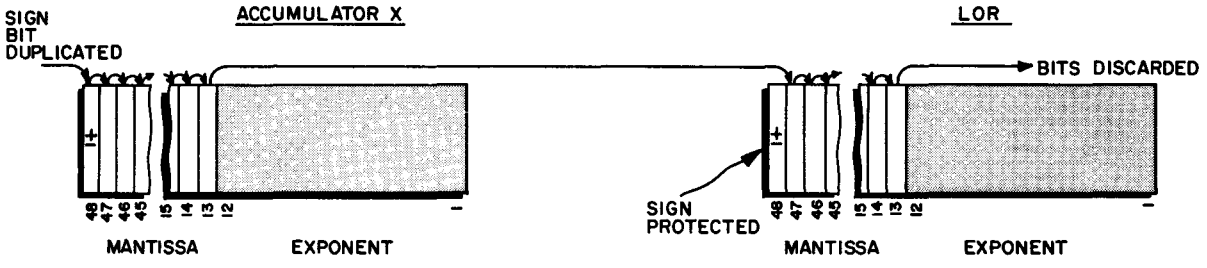
M=5: RIGHT, ARITHMETIC, SINGLE-PRECISION SHIFT



M=6: RIGHT, ROTATE, DOUBLE-PRECISION SHIFT



M=7: RIGHT, ARITHMETIC, DOUBLE-PRECISION SHIFT



TIMING

$9 + N_s/4$ cycles

REGISTERS AFTER OPERATION

AAR	BAR
A_p	B_p

NOTES

1. At the end of a shift operation, the exponents of (X) and (LOR) are zero.
2. In a single-precision shift, the mantissa portion of the previous contents of LOR is unaltered.

EXAMPLE

Perform a left, arithmetic, single-precision shift on accumulator 1. Shift by 12 bit positions.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

CARD NUMBER	TYPE	MAR	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
				BMS	11, 12	

SECTION VI

BINARY INTEGER ARITHMETIC INSTRUCTION

BINARY INTEGER MULTIPLY

FORMAT

BIM/A, B

FUNCTION

The four-character fields in main memory whose low-order characters are A and B are treated as 24-bit, twos-complement binary integers. The integers are multiplied together, and the product is stored in the field specified by the B address.

TIMING

$N_i + 20 + N_1/2$ cycles

REGISTERS AFTER OPERATION

AAR	BAR	LOR
A-4	B-4	unspecified

NOTES

1. If the product exceeds 24 bits, a multiply overflow indication is given and the low-order 24 bits are delivered to the field specified by the B address. Any high-order bits are lost.
2. The product is not shifted in any way.

EXAMPLE

Multiply the binary equivalent of 735_{10} by the binary equivalent of 899_{10}.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				INT1	DCW	#+8735	
2				INT2	DCW	#+8899	
3					BIM	INT1, INT2 product is delivered to INT2	

HONEYWELL
ELECTRONIC
DATA
PROCESSING
WELLESLEY HILLS,
MASSACHUSETTS 02181