File No. 1620/1710-13
Form A26-5708-0

IBM Systems Reference Library

IBM 1620 Special Features

This manual describes the special features that are available for the 1620 Data Processing System.

The format of the 1620 Reference Manual has been changed to conform to that of the Systems Reference Library. The original publication, A26-4500-2 and applicable newsletters have not been obsoleted.

This publication contains only the special features described in A26-4500-2. Minor changes have been made.

The Central Processing Unit is described in A26-5706.

Input/Output Units are described in A26-5707.

CONTENTS

IBM 1620 Data Processing System

The 1623 Core Storage Unit (Figure 1) provides the additional program and data storage needed for applications that require more than 20,000 storage positions.

Description

Two different 20,000-position modules of core storage are available. These expand the storage capacity of the 1620 from 20,000 positions to 40,000 positions with the Model 1 or 60,000 positions with the Model 2.

The programming and operating characteristics of the 1620 are not changed by additional core storage. Addressing is from 00000 to 39999 with the addition of the Model 1, and from 00000 to 59999 with the addition of the Model 2. The resulting storage is cyclincal in that 00000 follows the largest allowable address when incrementing. Conversely, the largest allowable address is the next lower address below 00000. No storage reservations for table arithmetic are required in additional core storage.

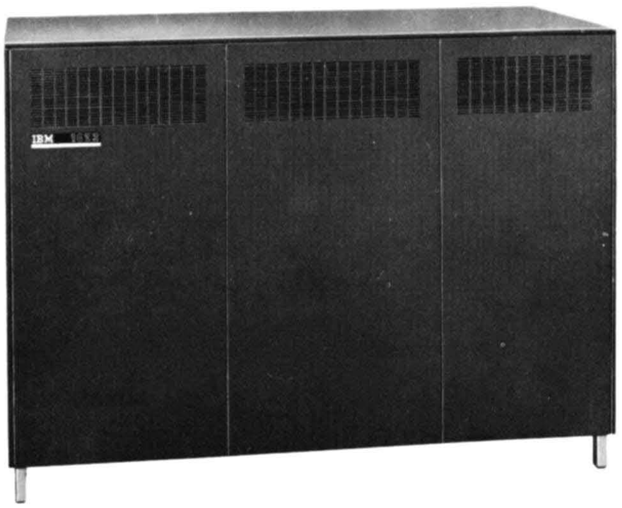


Figure 1. IBM 1623 Core Storage Unit

1623 Core Storage Cabinet

The 1623 cabinet is approximately the same in size as the 1620 console without the table top. There are no additional console controls. A power cord and signal cables 20 feet in length are provided for connecting the additional core storage cabinet to the 1620.

Checking

An invalid address is detected in MAR, and the MAR Check indicator is turned on when the digits listed below are in the high-order position (P_2 or Q_7) of either the P or Q address:

Core Storage	Error Digits in P_2 or Q_7
One module (00000-39999)	4, 5, 6, 7, 8, 9
Both Modules (00000-59999)	6, 7, 8, 9

Thus, a 40,000 position storage unit cannot have a valid address greater than 39999; and a high-order digit greater than 3 (for example, 4 for address 40000) is invalid. Similarly, an address of 60,000 or greater is invalid for a 60,000-position unit, for which the largest allowable address is 59999.

AUTOMATIC DIVISION

Automatic division simplifies programming and increases the processing speed of division problems by two to four times that of programmed routines. Only one command need be given. Four commands are provided, however, to facilitate positioning of the dividend and divisor in core storage. There are no practical limitations placed upon the size of the dividend, divisor, or quotient.

A quotient and remainder of 20 digits are developed in the product area (00080-00099). When the quotient plus the remainder exceeds 20 digits, core storage positions lower than 00080 (00079, 00078, etc.) must be reset to zeros by programming. One additional position should also be cleared to allow for a possible overdraw. For example, if 25 positions are required for the quotient and remainder, 00074-00079 would have to be reset to zeros before the divide command was given.

The four instructions provided with the divide feature are:

Load Dividend (LD-28)
Load Dividend Immediate (LDM-18)
Divide (D-29)
Divide Immediate (DM-19)

The formula for computing the total execution time follows the description of each instruction.

LOAD DIVIDEND (LD-28)

Description. The dividend must be stored in the product area before a Divide command is given. The Load Dividend instruction may be used to satisfy this requirement.

The product area (00080-00099) is automatically reset to zeros. The dividend (Q address) is transmitted to the product area (P address), beginning at the low-order dividend digit and terminating at the flag bit marking the high-order position of the dividend field. The P address is 00099 minus the number of zero positions desired to the right of the dividend.

The algebraic sign of the dividend is automatically placed in location 00099, regardless of where the low-order dividend digit is placed by the P address. A flag bit automatically marks the high-order digit of the dividend.

Example: Two Load Dividend instructions and one Load Dividend Immediate instruction are shown in Figure 2.

1. The Load Dividend instruction,

   28 00096 00650,

   causes the low-order position of the dividend to be placed at 00096. The sign (minus) is stored at 00099.

2. The Load Dividend instruction,

   28 00099 00650,

   causes the low-order position of the dividend to be placed at 00099. The sign (plus) is stored at 00099.

Instruction	Data at Core Storage Address 00650	Description	00080	00081	00082	. . . .	00092	00093	00094	00095	00096	00097	00098	00099
(1) 28 00096 00650	$\bar{2}136\bar{5}$	Load Dividend	0	0	0	. . . .	$\bar{2}$	1	3	6	5	0	0	$\bar{0}$
(2) 28 00099 00650	$\bar{0}1234$	Load Dividend	0	0	0	. . . .	0	0	0	$\bar{0}$	1	2	3	4
(3) 18 00098 $\bar{0}0650$	56789	Load Dividend Immediate	0	0	0	. . . .	0	0	$\bar{0}$	0	6	5	0	0

Figure 2. Load Dividend Instructions

3. The Load Dividend Immediate instruction,

$$18\ 00098\ \bar{0}0650,$$

causes the low-order position of the dividend (the Q part of the instruction) to be placed in the field beginning at 00098. The sign (plus) is stored at 00099.

Execution Time. $T = 400 + 40D_n$, where D_n equals the number of digits in the dividend.

LOAD DIVIDEND IMMEDIATE (LDM-18)

Description. The description for Load Dividend applies except that the data in the Q part of the instruction is transmitted to the P address.

Execution Time. Same as a Load Dividend (LD-28).

DIVIDE (D-29)

Description. The divisor (Q address) is successively subtracted from the dividend. The P address of the Divide instruction positions the divisor for the first subtraction from the high-order positions of the dividend, as in manual division. The P address is determined by subtracting the number of digits in the quotient from 100.

Examples: Problem 1: $\bar{4}906 \div \bar{2}3 = \bar{0}213$ and a remainder of $\bar{0}7$. Figure 3 shows the manner in which the 1620 solves this problem.

Problem 2: -212 ($\bar{2}1\bar{2}$) $\div$ $\bar{2}4 = -8.83$ ($\bar{0}088\bar{3}$) and a remainder of $\bar{0}8$. Figure 4 shows show the 1620 solves this problem.

As illustrated in these examples, each subtraction without overdraw causes the quotient digit to be increased by 1. Quotient digits are developed in the units position of the Multiplier/Quotient register. An overdraw initiates a correction cycle (the divisor is added once), and the next subtraction occurs one place to the right.

The first (high-order) quotient digit is stored at the address equal to the P address of the Divide instruction minus the length of the divisor. A flag bit is generated and stored with the first quotient digit. Subsequent quotient digits are stored to the right of the last-stored quotient digit. Division is terminated, after the last quotient digit is developed by subtractions, with the units position of the divisor at 00099.

The quotient and remainder replace the dividend in the product area. The address of the quotient is 00099 minus the length of the divisor. The algebraic sign of the quotient (determined by the signs of the dividend and divisor) is automatically placed in the low-order position of the quotient. The address of the remainder is 00099. A flag bit is automatically placed in the high-order position. The remainder has the sign of the dividend and the same number of digits as the divisor.

The High/Positive indicator is on if the quotient is positive and not zero; the Equal/Zero indicator is on if the quotient is zero. Neither indicator is on if the quotient is negative.

The quotient must be at least two digits in length; one position is required for the sign and one for the field mark (flag bit).

Execution Time. $T = 160 + 520D_vQ_t + 740Q_t$. D_v and Q_t equal the number of digits in the divisor and quotient, respectively. The formula assumes an average quotient digit of 4.5. If a Load Dividend or Load Dividend Immediate instruction is used, the divide operation execution time may be considered as the total time for both the Load Dividend and Divide instructions.

$$T = 500 + 40D_n + 520D_vQ_t + 740Q_t$$

DIVIDE IMMEDIATE (DM-19)

Description. The description of Divide (D-29) applies except that the data in the Q part of the instruction is used as the divisor.

Execution Time. Same as Divide (D-29).

DECIMAL POINT LOCATION

The computer is unaware of decimal points, except for Automatic Floating-Point Operations (Special Feature). Decimal point location for any given divide calculation is easily determined by simply subtracting the number of decimal digits in the divisor from the number of decimal digits in the dividend. The result is the number of decimal digits in the quotient. For example, if the divisor and dividend values in problem 2, Figure 4 are 2.4 and 21.200, respectively, the quotient value is 008.83 (3 - 1 = 2). Note that the original dividend, 21.4 became 21.400 as a result of its placement by the Load Dividend instruction. Thus, the number of dividend decimal digits must include the zeros to the right of the loaded dividend.

Instruction	Data at Core Storage Addresses		Description	00092	00093	00094	00095	00096	00097	00098	00099
	00500	00600									
28 00099 00500	$\bar{4}906$	$\bar{2}3$	Load dividend	0	0	0	0	$\bar{4}$	9	0	6
29 00096 00600			Subtract divisor			−	2	3			
			Overdraw			9	8	1			
			Add divisor back to correct overdraw.			+	2	3			
						0	0	4			
			Store first (high-order) digit of quotient (0) and flag bit	0	0	$\boxed{\bar{0}}$	0	4	9	0	6
			Subtract divisor one place to the right				−	2	3		
			No overdraw					2	6		
			Subtract divisor				−	2	3		
			No overdraw				0	0	3		
			Subtract divisor				−	2	3		
			Overdraw				9	8	0		
			Add divisor back to correct overdraw				+	2	3		
							0	0	3		
			Store second digit of quotient (2)	0	0	$\bar{0}$	$\boxed{2}$	0	3	0	6
			Subtract divisor one place to the right					−	2	3	
			No overdraw					0	0	7	
			Subtract divisor					−	2	3	
			Overdraw					9	8	4	
			Add back divisor to correct overdraw					+	2	3	
								0	0	7	
			Store third digit of quotient (1)	0	0	$\bar{0}$	2	$\boxed{1}$	0	7	6
			Subtract divisor one place to the right						−	2	3
			No overdraw						0	5	3
			Subtract divisor						-	2	3
			No overdraw						0	3	0
			Subtract divisor						−	2	3
			No overdraw						0	0	7
			Subtract divisor						−	2	3
			Overdraw						9	8	4
			Add back divisor to correct overdraw						+	2	3
									0	0	7
			Store fourth digit of quotient (3) and flag bit, if negative. Operation stops with quotient ($\bar{0}213$) and remainder ($\bar{0}7$) in product area.	0	0	$\bar{0}$	2	1	$\boxed{3}$	$\bar{0}$	7

Figure 3. Divide, Problem 1

Instruction	Description	00650	00500	00090	00091	00092	00093	00094	00095	00096	00097	00098	00099
	Data	$\bar{2}4$	$\bar{2}1\bar{2}$										
LD 28 00097 00500	Reset 00080 - 00099 to zeros. Transmit dividend to 00097. Dividend sign to 00099.			0	0	0	0	0	$\bar{2}$	1	2	0	$\bar{0}$
D 29 00095 00650	Subtract divisor from dividend starting at 00095.						–	2	4				
	Overdraw						9	7	8				
	Correction						+	2	4				
							0	0	2				
	Store first quotient digit (0) and flag bit			0	0	0	$\boxed{\bar{0}}$	0	2	1	2	0	$\bar{0}$
	Subtract one place to the right							–	2	4			
	Overdraw							9	9	7			
	Correction							+	2	4			
								0	2	1			
	Store 2nd quotient digit (0)			0	0	0	$\bar{0}$	$\boxed{0}$	2	1	2	0	$\bar{0}$
	Subtract one place to the right								–	2	4		
	Successful subtraction								1	8	8		
	7 more successful subtractions (7 x 24 = 168)								–1	6	8		
									0	2	0		
									–	2	4		
	Overdraw								9	9	6		
	Correction								+	2	4		
									0	2	0		
	Store quotient digit (8)			0	0	0	$\bar{0}$	0	$\boxed{8}$	2	0	0	$\bar{0}$
	8 successful subtractions (8 x 24 = 192)								–	1	9	2	
	(Overdraw and correction not shown)									0	0	8	
	Store quotient digit (8)			0	0	0	$\bar{0}$	0	8	$\boxed{8}$	0	8	$\bar{0}$
	3 successful subtractions (3 x 24 = 72)										–	7	2
												0	8
											–	2	4
	Overdraw										9	8	4
	Correction										+	2	4
											0	0	8
	Store quotient digit (3)			0	0	0	$\bar{0}$	0	8	8	$\boxed{3}$	0	$\bar{8}$
	Store flag over high-order position of remainder. Sign of quotient over units position (00099 - length of divisor).			0	0	0	$\bar{0}$	0	8	8	$\bar{3}$	$\bar{0}$	$\bar{8}$

Figure 4. Divide, Problem 2

INCORRECT DIVISOR POSITIONING

The following error conditions are caused by an incorrect P address in the Divide instruction:

Overflow. As illustrated in Figure 5, an incorrectly positioned divisor can cause more than nine successful subtractions and an incorrect quotient. The divide operation is terminated, the Arithmetic Check indicator and light are turned on, but processing does not stop unless the Overflow Check switch is set to STOP. Note the absence of a field-length flag in position 00095 when division is terminated. The flag is not placed automatically because the first quotient digit, which normally causes the flag bit to be generated and stored, is not achieved.

If, after a division overflow, the field remaining in the product area is to be used for further operations, the program must provide for a flag to be set in the desired position.

Loss of One or More High-Order Digits of the Dividend. The high-order digit of the dividend is assumed by the 1620 to be one position to the left of the high-order digit of the divisor. Figure 6 shows how the high-order digits of the dividend are lost if the divisor is positioned too far to the right. Processing continues with no indication of an incorrect quotient.

Incorrect Termination. If the P address is less than 10000, i.e., between 00100 and 09999, the divide operation will terminate when a subtraction

Instruction	Description	00650	00090	00091	00092	00093	00094	00095	00096	00097	00098	00099
		$\bar{2}$ 1	0	0	0	0	0	$\bar{2}$	1	2	0	$\bar{0}$
D 29 00097 00650	Successful subtraction No. 1							−	2	1		
								1	9	1		
	" " No. 2							−	2	1		
								1	7	0		
	" " No. 3							−	2	1		
								1	4	9		
	" " No. 4							−	2	1		
								1	2	8		
	" " No. 5							−	2	1		
								1	0	7		
	" " No. 6							−	2	1		
								0	8	6		
	" " No. 7							−	2	1		
								0	6	5		
	" " No. 8							−	2	1		
								0	4	4		
	" " No. 9							−	2	1		
								0	2	3		
	" " No. 10							−	2	1		
			0	0	0	0	0	0	0	2	0	$\bar{0}$

Figure 5. Divide Overflow

Instruction	Description	00650	00095	00096	00097	00098	00099
29 00098 00650	Divide (Incorrect P Address)	$\bar{1}9$	$\bar{2}$	0	2	3	$\bar{0}$
				–	1	9	
				0	0	4	
				–	1	9	
				9	8	5	
				+	1	9	
				0	0	4	
			$\bar{2}$	$\boxed{\bar{1}}$	0	4	0
					–	1	9
					0	2	1
					–	1	9
					0	0	2
					–	1	9
					9	8	3
					+	1	9
					0	0	2
			$\bar{2}$	$\bar{1}$	$\boxed{2}$	$\bar{0}$	2

Figure 6. Division Error, Incorrect Programming

occurs at 0XX99. This, in effect, restricts the size of the dividend to 10,020 digits, if only 20,000 positions of core storage are installed.

SUMMARY OF AUTOMATIC DIVISION RULES

1. Load Dividend (LD-28 or LDM-18)
   a. P address = 00099 minus the number of zeros desired to the right of the units position of the dividend.
   b. Q address = core storage address of the dividend.
2. Divide (D-29 or DM-19)
   a. P address = 00100 minus the length of the quotient. The quotient length is 100 minus the P address.
   b. Q address = core storage address of the divisor.
3. Quotient address = 00099 minus the length of the divisor.
4. Quotient length = 100 minus P address
5. Remainder address = 00099.
6. Sign of quotient: determined by the algebraic signs of the dividend and divisor.
7. Sign of remainder: same as that of the dividend.
8. Decimal point location: the number of dividend decimal digits minus the number of divisor decimal digits equals the number of quotient decimal digits.

AUTOMATIC FLOATING-POINT OPERATIONS

This special feature provides the 1620 with the ability to do floating-point arithmetic, using floating-point instructions instead of program subroutines.

The use of automatic floating-point operations can result in a 50 to 100 per cent increase in the computing power of the 1620 CPU, depending on the amount of floating-point computations required. Also, up to 15 per cent of the basic 1620 core-storage capacity can be saved through the elimination of subroutines and call sequence instructions associated with Floating Add, Floating Subtract, Floating Multiply, and Floating Divide.

The Automatic Division special feature is a prerequisite to the installation of Automatic Floating-Point Operations.

Floating-Point Arithmetic

Scientific and engineering computations frequently involve lengthy and complex calculations in which it is necessary to manipulate numbers that may vary widely in magnitude. To obtain a meaningful answer, problems of this type usually require that as many significant digits as possible be retained during calculation and that the decimal point always be properly located. When applying such problems to a computer; several factors must be taken into consideration, the most important of which is decimal point location.

Generally speaking, a computer does not recognize the decimal point present in any quantity used during the calculation. Thus, a product of 414154 will result regardless of whether the factors are 9.37 x 44.2, 93.7 x .442, or 937 x 4.42, etc. It is the programmer's responsibility to be cognizant of the decimal point location during and after the calculation and to arrange the program accordingly. In a floating add operation, for example, the decimal point of all numbers must be lined up to obtain the correct sum. To facilitate this arrangement, the programmer must shift the quantities as they are added. In the manipulation of numbers that vary greatly in magnitude, the resulting quantity could conceivably exceed allowable working limits.

The processing of numbers expressed in ordinary form, e.g., 427.93456, 0.0009762, 5382, -623.147, 3.1415927, etc., can be accomplished on a computer only by extensive analysis to determine the size and range of intermediate and final results. This analysis and subsequent number scaling frequently takes longer than does the actual calculation. Furthermore, number scaling requires complete and accurate information as to the boundaries of all numbers that come into the computation (input, intermediate, output). Since it is not always possible to predict the size of all numbers in a given calculation, analysis and number scaling are sometimes impractical.

To alleviate this programming problem, a system must be employed in which information regarding the magnitude of all numbers accompanies the quantities in the calculation. Thus, if all numbers are represented in some standard, predetermined format which instructs the computer in an orderly and simple fashion as to the location of the decimal point, and if this representation is acceptable to the routine doing the calculation, then quantities which range from minute fractions having many decimal places to large whole numbers having many integer places can be handled. The arithmetic system most commonly used, in which all numbers are expressed in a format having the above feature, is called "floating-point arithmetic."

The notation used in floating-point arithmetic is basically an adaptation of the scientific notation widely used today. In scientific work, very large or very small numbers are expressed as a number, between one and ten, times a power of ten. Thus 427.93456 is written as 4.2793456×10^2 and 0.0009762 as 9.762×10^{-4}. In the 1620 floating-point arithmetic system, the range of numbers is modified to extend between .1 and .99999999, that is, the decimal point of all numbers is placed to the left of the high-order (leftmost) nonzero digit. Hence, all quantities may be thought of as a decimal fraction times a power of ten (e.g., 427.93456 as $.42793456 \times 10^3$ and 0.0009762 as $.97620000 \times 10^{-3}$) where the fraction is called the mantissa, and the power of ten, used to indicate the number of places the decimal point was shifted, the exponent. In addition to the advantages inherent in scientific notation, the use of floating-point numbers during processing eliminates the necessity of analyzing the operations to determine the positioning of the decimal point in intermediate and final results, since the decimal point is always immediately to the left of the high-order nonzero digit in the mantissa.

1620 Automatic Floating-Point Operations

In 1620 Automatic Floating-Point Operations, a floating-point number is a field consisting of a variable length mantissa and a 2-digit exponent. The exponent is in the two low-order positions of the field, and the mantissa is in the remaining high-order positions, as shown:

$$\bar{M}. . . . M\bar{E}E$$

The mantissa must have a minimum of two digits and can have a maximum of 100 digits. However, when two fields are operands (quantities being added, subtracted, multipled, divided), they must have mantissas of the same length. The extremity of the field is marked by a flag over the high-order digit.

The exponent is established on the premise that the mantissa is less than 1.0 and equal to or greater than 0.1. The exponent is always two digits and has a range of -99 to +99. The length of the exponent field is defined by a flag over the high-order (tens) digit

The mantissa and the exponent each have an algebraic sign represented by the presence (negative) or absence (positive) of a flag over the units position. A floating-point number with a negative mantissa and a negative exponent is represented as follows:

$$\bar{M}. . . . \bar{M}\bar{E}\bar{E}$$

Sign control of the results of all computations is maintained according to the standard rules of arithmetic operations.

Eight floating-point instructions are provided: four are for arithmetic computations -- Floating Add, Floating Subtract, Floating Multiply, and Floating Divide; three are used to control field size and location -- Floating Shift Right, Floating Shift Left, and Transmit Floating. The eighth instruction provides for Branch and Transmit in floating-point operations. All instructions are in the 1620 format of a 2-digit Op code, 5-digit P address, and 5-digit Q address.

As an aid to the programmer or operator in checking program logic and computation results, the operation of the computer in aligning decimal points, normalizing results, etc., is described with each instruction. These operations are automatic and need not be programmed. Of particular note is Floating Divide, which requires only one instruction; the dividend is positioned, division is accomplished, and the quotient is transmitted to the P field without further command.

In descriptions of instructions and operations, the following symbols are used for clarity and brevity:

Mp	=	mantissa of the field at the P address (P)
Mq	=	mantissa of the field at the Q address (Q)
Ep	=	exponent of the field at the P address
Eq	=	exponent of the field at the Q address
L	=	number of digits in the mantissa
d	=	Ep - Eq

In all floating-point numbers, the decimal point is assumed to be at the left of the high-order digit, which must not be zero. Such a number is referred to as "normalized." When a number has one or more high-order zeros, it is considered to be "unnormalized." An unnormalized number resulting from a floating-point computation is normalized automatically, but unnormalized terms are not recognized as such when entered as data. They will be processed, but correct results cannot be assured. Therefore, it is necessary that all data be entered in normalized form. For example, the number $\bar{0}68234940\bar{5}$ should be entered as $\bar{6}8234940\bar{0}4$, assuming the fixed-point number is 6823.494, and an 8-digit mantissa is required.

With the exception of Floating Shift Right and Floating Shift Left, the P address and Q address of floating point fields are the addresses of the low-order positions of the exponents.

FLOATING ADD (FADD-01)

Description. Mq is added to Mp and the result replaces Mp. Mq and Eq are not altered in core storage. Dependent on L and the value of d, the appropriate field is shifted to align decimal points before addition is performed. If d = 0, no shift is made (Figure 7).

If d is greater than zero and less than L, in effect, Mq is shifted d positions to the right before being added to Mp. The number of low-order digits of Mq equal to d are truncated as the shift is made (Figure 8). If d is less than zero, and the absolute

Core Storage Locations 01590 ⟷ 01599				Instruction			Core Storage Locations 01590 ⟷ 01599			
Before							After			
Mp	Ep	Mq	Eq	OP	P	Q	Mp	Ep	Mq	Eq
$\bar{1}$ 2 3	$\bar{0}$ 4	$\bar{7}$ 8 9	$\bar{0}$ 4	0 1	0 1 5 9 4	0 1 5 9 9	$\bar{9}$ 1 2	$\bar{0}$ 4	$\bar{7}$ 8 9	$\bar{0}$ 4

Figure 7. Addition Without Mp or Mq Shift

Core Storage Locations 01590 ⟷ 01599				Instruction			Core Storage Locations 01590 ⟷ 01599			
Before							After			
Mp	Ep	Mq	Eq	OP	P	Q	Mp	Ep	Mq	Eq
$\bar{1}$ 2 3	$\bar{0}$ 2	$\bar{7}$ 8 9	$\bar{0}$ 1	0 1	0 1 5 9 4	0 1 5 9 9	$\bar{2}$ 0 1	$\bar{0}$ 2	$\bar{7}$ 8 9	$\bar{0}$ 1

Figure 8. Mq Shifted Right to Align Decimal Points

value of d, is less than L, Mp is shifted |d| positions to the right before Mq is added to Mp. The number of low-order digits of Mp equal to |d| are truncated as the shift is made. Eq replaces Ep (Figure 9). If d is plus and equal to or larger than L, Mp is above the range of Mq and no addition is performed (Figure 10). If d is less than zero and |d| is equal to or greater than L, Mq is above the range of Mp, and no addition is performed. Mq replaces Mp, and Eq replaces Ep (Figure 11).

Core Storage Locations 01590 ⟷ 01599				Instruction			Core Storage Locations 01590 ⟷ 01599			
Before							After			
Mp	Ep	Mq	Eq	OP	P	Q	Mp	Ep	Mq	Eq
$\bar{1}$ 2 3	$\bar{0}$ 1	$\bar{7}$ 8 9	$\bar{0}$ 2	0 1	0 1 5 9 4	0 1 5 9 9	$\bar{8}$ 0 1	$\bar{0}$ 2	$\bar{7}$ 8 9	$\bar{0}$ 2

Figure 9. Mp Shifted Right to Align Decimal Points

Core Storage Locations 01590 ⟷ 01599				Instruction			Core Storage Locations 01590 ⟷ 01599			
Before							After			
Mp	Ep	Mq	Eq	OP	P	Q	Mp	Ep	Mq	Eq
$\bar{1}$ 2 3	$\bar{0}$ 5	$\bar{7}$ 8 9	$\bar{0}$ 2	0 1	0 1 5 9 4	0 1 5 9 9	$\bar{1}$ 2 3	$\bar{0}$ 5	$\bar{7}$ 8 9	$\bar{0}$ 2

Figure 10. Mp Above Range of Mq

Core Storage Locations 01590 ⟷ 01599				Instruction			Core Storage Locations 01590 ⟷ 01599			
Before							After			
Mp	Ep	Mq	Eq	OP	P	Q	Mp	Ep	Mq	Eq
$\bar{1}$ 2 3	$\bar{0}$ $\bar{1}$	$\bar{7}$ 8 9	$\bar{0}$ 3	0 1	0 1 5 9 4	0 1 5 9 9	$\bar{7}$ 8 9	$\bar{0}$ 3	$\bar{7}$ 8 9	$\bar{0}$ 3

Figure 11. Mq Above Range of Mp

After addition has been completed, the number of Mp digits is checked to determine if it exceeds L. If so, this is an overflow condition; the low-order digit of Mp is truncated, and the mantissa is shifted one position to the right. A one is entered in the high-order position of the mantissa, and a one is added to Ep (Figure 12). When an overflow does not exist, Mp is scanned for zeros beginning with the high-order position. High-order zeros are counted (z), and Mp is shifted z positions to the left; vacated positions are set to zeros. Flag bits in Mp are not altered or moved. Eq - z replaces Ep (Figure 13).

Core Storage Locations 01590 ⟷ 01599				Instruction			Core Storage Locations 01590 ⟷ 01599			
Before							After			
Mp	Ep	Mq	Eq	OP	P	Q	Mp	Ep	Mq	Eq
$\bar{9}$ 8.7	$\bar{0}$ 4	$\bar{4}$ 5 6	$\bar{0}$ 4	0 1	0 1 5 9 4	0 1 5 9 9	$\bar{1}$ 4 4	$\bar{0}$ 5	$\bar{4}$ 5 6	$\bar{0}$ 4

Figure 12. Mantissa Overflow, Number Normalized

Core Storage Locations 01590 ⟷ 01599				Instruction			Core Storage Locations 01590 ⟷ 01599			
Before							After			
Mp	Ep	Mq	Eq	OP	P	Q	Mp	Ep	Mq	Eq
$\bar{1}$ 2 3	$\bar{0}$ 1	$\bar{1}$ 1 $\bar{9}$	$\bar{0}$ 1	0 1	0 1 5 9 4	0 1 5 9 9	$\bar{4}$ 0 0	$\bar{0}$ $\bar{1}$	$\bar{1}$ 1 $\bar{9}$	$\bar{0}$ 1

Figure 13. High-Order Zeros, Number Normalized

Execution Time. T (average) = 400 + 100 L. If the result is recomplemented, add 80L.

FLOATING SUBTRACT (FSUB-02)

Description. The floating subtract operation is the same as the floating add operation except that sign control procedures for Mq are reversed.

Execution Time. Same as Floating Add (FADD-01).

FLOATING MULTIPLY (FMUL-03)

Description. Mp is multiplied by Mq, and the result replaces Mp. Ep is added to Eq, and the sum replaces Ep. Mp and Ep are normalized, as required, after multiplication. Mq and Eq are not altered in core storage. The product is formed in the product area, beginning at 00099 and extending through lower-numbered core storage positions to 00100 − 2L. The product area, 00080-00099, is cleared automatically prior to multiplication. If L is greater than 10, the program must provide for clearing positions 00100 − 2L through 00079. After multiplication, the digit at position 00100 − 2L is tested for zero. If the digit is other than a zero, the field at 00099 − L replaces Mp (Figure 14). If the digit

Core Storage Locations 01590 $\longleftrightarrow$ 01599				Instruction			Core Storage Locations 01590 $\longleftrightarrow$ 01599			
Before							After			
Mp	Ep	Mq	Eq	OP	P	Q	Mp	Ep	Mq	Eq
$\bar{7}$ 8 9	$\bar{0}$ 3	$\bar{4}$ 5 6	$\bar{0}$ $\bar{1}$	0 3	0 1 5 9 4	0 1 5 9 9	$\bar{3}$ 5 9	$\bar{0}$ 2	$\bar{4}$ 5 6	$\bar{0}$ $\bar{1}$

Figure 14. Product Equal to 2L

tested is a zero, the field at 00100 — L replaces Mp and Ep + Eq — 1 replaces Ep (Figure 15).

Execution Time. T (average) = 1120 + 80L + $168L^2$.

Core Storage Locations 01590 $\longleftrightarrow$ 01599				Instruction			Core Storage Locations 01590 $\longleftrightarrow$ 01599			
Before							After			
Mp	Ep	Mq	Eq	OP	P	Q	Mp	Ep	Mq	Eq
$\bar{1}$ 2 3	$\bar{0}$ 2	$\bar{4}$ 5 6	$\bar{0}$ 4	0 3	0 1 5 9 4	0 1 5 9 9	$\bar{5}$ 6 0	$\bar{0}$ 5	$\bar{4}$ 5 6	$\bar{0}$ 4

Figure 15. Product Less then 2L

FLOATING DIVIDE (FDIV-09)

Description. Mp is divided by Mq, and the quotient replaces Mp. Eq is deducted from Ep, and the result replaces Ep. Mp and Ep are normalized, as required, after division, Mq and Eq are not altered in core storage. The quotient and remainder are developed in the product area, beginning at 00099 and extending through lower-numbered core storage positions to 00100 − 2L. The product area, 00080-00099, is cleared automatically prior to division. If L is greater than 10, the program must provide for clearing positions 00100 − 2L through 00079. Prior to division, the absolute values of Mp and Mq are compared. If Mp is equal to or greater than Mq, Mp is transmitted to 00100 − L, and division is performed, starting at 00100 − L, according to the procedure for automatic division. The quotient at 00099 − L replaces Mp, and Ep - Eq + 1 replaces Ep (Figure 16). If Mp is less than Mq, Mp is transmitted to 00099 – L; division starts in 00100 −L, and proceeds according to the procedure for automatic division. The quotient at 00099 − L replaces Mp, and Ep - Eq replaces Ep (Figure 17).

Core Storage Locations 01590 $\longleftrightarrow$ 01599				Instruction			Core Storage Locations 01590 $\longleftrightarrow$ 01599			
Before							After			
Mp	Ep	Mq	Eq	OP	P	Q	Mp	Ep	Mq	Eq
$\bar{7}$ 8 9	$\bar{0}$ 4	$\bar{1}$ 2 3	$\bar{0}$ 1	0 9	0 1 5 9 4	0 1 5 9 9	$\bar{6}$ 4 1	$\bar{0}$ 4	$\bar{1}$ 2 3	$\bar{0}$ 1

Figure 16. Divisor Equal to or Less than Dividend

Core Storage Locations 01590 $\longleftrightarrow$ 01599				Instruction			Core Storage Locations 01590 $\longleftrightarrow$ 01599			
Before							After			
Mp	Ep	Mq	Eq	OP	P	Q	Mp	Ep	Mq	Eq
$\bar{1}$ 2 3	$\bar{0}$ 1	$\bar{7}$ 8 9	$\bar{0}$ 4	0 9	0 1 5 9 4	0 1 5 9 9	$\bar{1}$ 5 5	$\bar{0}$ $\bar{3}$	$\bar{7}$ 8 9	$\bar{0}$ 4

Figure 17. Divisor Greater than Dividend

Division by zero causes the Arithmetic Check indicator (14) to be turned on. Mp is not altered, but Ep is replaced by Ep - Eq.

Execution Time. T = 880 + 940L + $520L^2$. The formula is based on an average quotient digit of 4.5.

FLOATING SHIFT RIGHT (FSR-08)

Description. The field at the Q address (the portion of the mantissa to be retained) is shifted right to the location specified by the P address. The exponent is not moved or altered. The effect of this instruction is to shrink the mantissa by shifting it to the right and truncating the low-order digits. The P address is normally the units position of the mantissa; the digit at the Q address becomes the new low-order digit of the mantissa. Vacated high-order positions are set to zeros. An existing flag bit at the P address is retained for algebraic sign; the field flag bit is transmitted with the high-order digit of the Q field and terminates the operation (Figure 18).

Execution Time. T = 200 +40L.

Core Storage Locations 01590 $\longleftrightarrow$ 01599				Instruction			Core Storage Locations 01590 $\longleftrightarrow$ 01599			
Before							After			
Mp	Ep	Mq	Eq	OP	P	Q	Mp	Ep	Mq	Eq
0 $\bar{1}$ 2	$\bar{0}$ 2	$\bar{7}$ 8 9	$\bar{0}$ 5	0 8	0 1 5 9 7	0 1 5 9 6	0 $\bar{1}$ 2	$\bar{0}$ 2	0 $\bar{7}$ 8	$\bar{0}$ 5

Figure 18. Floating Shift Right

FLOATING SHIFT LEFT (FSL-05)

Description. The field at the Q address, which is the low-order position of the mantissa, is shifted left so that the high-order digit is moved to the location specified by the P address. The exponent is not moved or altered. The effect of this instruction is to expand the mantissa by shifting it to the left and filling the vacated positions with zeros. It is important to note that the Q address is the low-order position of the field moved, and the P address is the high-order position of the resulting field. An existing flag bit at the Q address is retained for algebraic sign; the field flag bit is transmitted with the high-order digit of the Q field (Figure 19).

Core Storage Locations 01590 ⟷ 01599				Instruction			Core Storage Locations 01590 ⟷ 01599			
Before							After			
Mp	Ep	Mq	Eq	OP	P	Q	Mp	Ep	Mq	Eq
$\overline{1}$ 2 3	$\overline{0}$ 2	0 $\overline{7}$ $\overline{8}$	$\overline{0}$ 5	0 5	0 1 5 9 5	0 1 5 9 7	$\overline{1}$ 2 3	$\overline{0}$ 2	$\overline{7}$ 8 $\overline{0}$	$\overline{0}$ 5

Figure 19. Floating Shift Left

If the mantissa is expanded to a length greater than 2L, any extraneous flag bits in core storage positions between the old high-order position and the new low-order position of the mantissa must be cleared before the FSL instruction is given. Therefore, if Q - P is equal to or greater than 2L, locations P + L through Q - L must be free of flags.

Contrary to other instructions in the floating-point series, FSL is executed in the transmit record manner of transmitting individual digits in the high-order to low-order sequence. After the units digit has been transmitted, the positions of the expanded mantissa are set to zero, in ascending core storage location sequence. After each position is set to zero, the succeeding position is checked for a flag bit. If the fraction is positive, the flag bit is assumed to be the high-order position of the exponent and the operation stops without altering the flag bit position. If the fraction is negative, the flag bit is assumed to be the units position of the fraction, and a negative zero is inserted in the units position before the operation stops. Thus, a flag bit detected prior to the previous high-order position of the mantissa stops the operation and results in an incorrect mantissa.

For example, if P = 01590, Q = 01601, and L = 4, core storage locations 01590 through 01603, with an extraneous flag bit in 01596, appear as follows:

$$XXXXXX\overline{X}X\overline{M}MMM\overline{E}E$$

After transfer of the mantissa, but before the zero-fill operation, the core storage locations appear as follows (note that the flag bit in 01598 has been cleared):

$$\overline{M}MMMXX\overline{X}XMMMM\overline{E}E$$

Upon completion of the operation, the mantissa is incorrect, as follows:

$$\overline{M}MMM00\overline{X}XMMMM\overline{E}E$$

If 01596 had not contained a flag bit, the mantissa would have been expanded correctly, as follows:

$$\overline{M}MMM00000000\overline{E}E$$

Execution Time. T = 200 + 40L + 40L'. (L' = length mantissa is increased by shift.)

TRANSMIT FLOATING (TFL-06)

Description. The field at the Q address is transmitted to the location designated by the P address. Mq and Eq are not altered in core storage. The Q address is normally the low-order position of the exponent, and the operation is the same as the regular Transmit Field instruction (TF-26), except that flag bits in the three low-order positions are ignored as indications to terminate the transmittal. Beginning with the fourth low-order digit, a flag bit terminates the operation. All flag bits in the field are transmitted (Figure 20).

Execution Time. T = 240 + 40L.

Core Storage Locations 01590 ⟷ 01599				Instruction			Core Storage Locations 01590 ⟷ 01599			
Before							After			
Mp	Ep	Mq	Eq	OP	P	Q	Mp	Ep	Mq	Eq
$\overline{1}$ 2 3	$\overline{0}$ 2	$\overline{7}$ 8 9	$\overline{0}$ 5	0 6	0 1 5 9 4	0 1 5 9 9	$\overline{7}$ 8 9	$\overline{0}$ 5	$\overline{7}$ 8 9	$\overline{0}$ 5

Figure 20. Transmit Floating

BRANCH AND TRANSMIT FLOATING (BTFL-07)

Description. The address of the next instruction is saved in IR-2, and the field at the Q address is transmitted to the P address minus one. The instruction at the P address is the next one executed. Mq and Eq are not altered in core storage. The Q address is normally the low-order position of the exponent. The operation is the same as the regular Branch and Transmit instruction (BT-27), except that in the transmit function the three low-order position flags are ignored as indications to terminate the transmittal. Beginning with the fourth low-order position, a flag bit terminates the operation. All flag bits are transmitted.

Execution Time: T = 280 + 40L.

MANTISSA AND EXPONENT ANALYSIS

Zero Mantissa

When a floating-point computation results in a zero mantissa, a special floating-point zero is created in the form $\bar{0}0 \ldots 0\bar{9}\bar{9}$, which is the smallest positive quantity that can be represented (Figure 21). A zero

Core Storage Locations 01590 ⟷ 01599				Instruction			Core Storage Locations 01590 ⟷ 01599			
Before							After			
Mp	Ep	Mq	Eq	OP	P	Q	Mp	Ep	Mq	Eq
$\bar{7}$ 8 9	$\bar{0}$ 5	$\bar{7}$ 8 9	$\bar{0}$ 5	0 2	0 1 5 9 4	0 1 5 9 9	$\bar{0}$ 0 0	$\bar{9}$ $\bar{9}$	$\bar{7}$ 8 9	$\bar{0}$ 5

Figure 21. Zero Mantissa

mantissa causes the Equal/Zero indicator (12) to be turned on. Zeros entered as data should be in floating-point zero form. Zero quantities in other forms, e.g., $\bar{0}0 \ldots 0\bar{0}0$ will be processed, but results cannot be assured.

Indicators

The four indicators associated with automatic floating-point operations are represented by lights on the 1620 console. The light for each indicator is turned on when the corresponding indicator is turned on. The High/Positive and Equal/Zero lights are located in the Control Gates section of the console, and the Arithmetic Check and Exponent Check lights and Overflow switch are in the Indicator Displays and Switches section (Figure 22).

High/Positive (11). The High/Positive indicator and light are turned on when the mantissa resulting from a floating-point computation is greater than zero.

Equal/Zero (12). The Equal/Zero indicator and light are turned on to indicate a zero mantissa resulting from a floating-point computation.

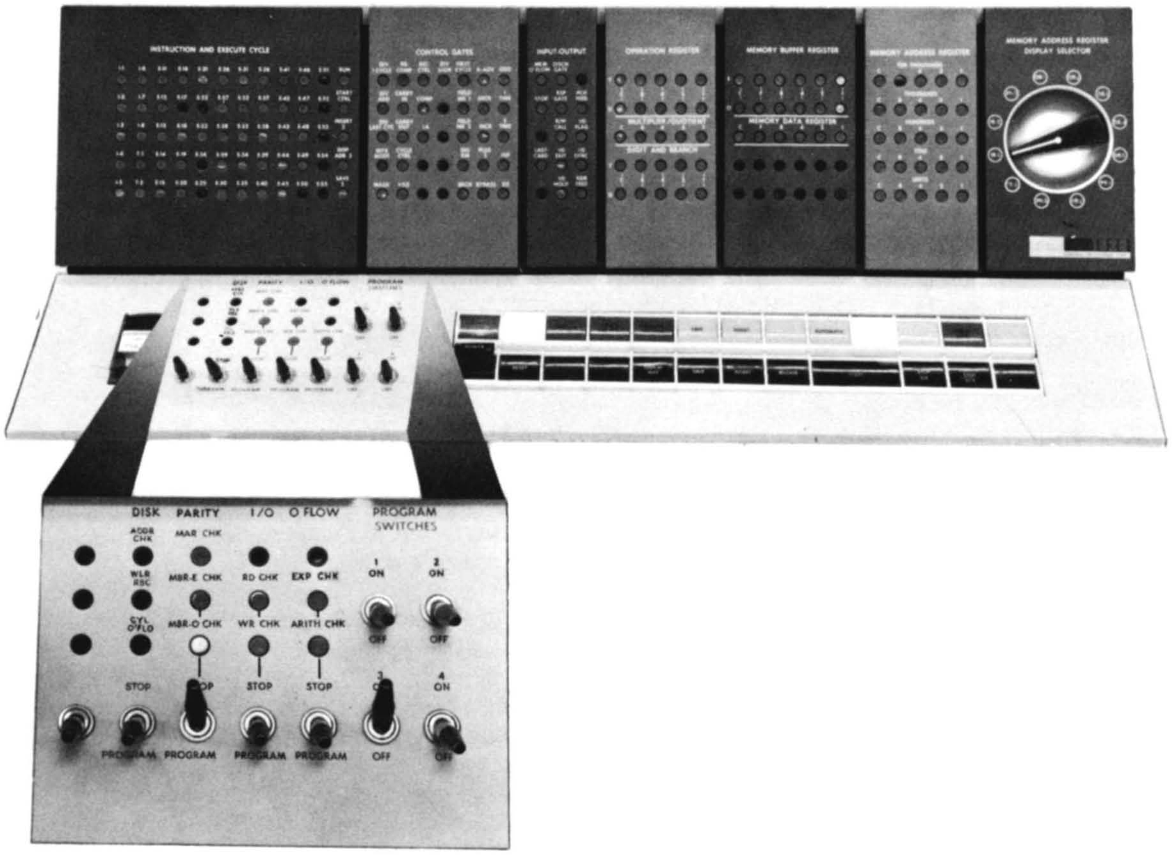


Figure 22. Indicators and Switches on 1620 Console

Arithmetic Check (14). During floating-point operations, the Arithmetic Check indicator is turned on when division is attempted by zero. Division by an unnormalized number may result in an incorrect quotient through incorrect positioning of the divisor.

Exponent Check (15). The Exponent Check indicator is turned on by exponent overflow or underflow.

Exponent Overflow

When an exponent greater than +99 is generated, the mantissa is set to nines. The sign is determined by the computed result that caused the overflow. The exponent is set to +99. This is the largest floating-point number ($\bar{9}9....9\bar{9}9$) that can be represented. If the generated mantissa is positive, the H/P indicator (11) is also turned on.

Exponent Underflow

When an exponent less than -99 is generated, the mantissa is set to plus zeros, and the exponent is set to -99. This is the smallest floating-point number ($\bar{0}0....0\bar{9}\bar{9}$) that can be represented. The E/Z indicator is also turned on.

An exponent underflow is not indicated when one or both operands are zero.

When the Exponent Check indicator (15) is turned on, program operation is controlled by the console Overflow Check switch, which is also connected to the Arithmetic Check indicator (14). The Exponent Check indicator (15) is turned off by programmed interrogation or by depression of the 1620 Reset key.

MARS Display Selector (1620 Console)

Operand Register 4. OR-4 is used to store and control the address of Eq.

Operand Register 5. OR-5 is used to store and control the address of Ep.

Counter Register 1. CR-1 is used to store the algebraic difference between Ep and Eq for determination of decimal alignment. It is also used to count high-order zeros when normalizing -- the contents of CR-1 are subtracted from Ep.

Indirect Addressing saves program steps and computer time by providing a direct method of address modification. Its primary use is in programs where multiple instructions have the same address, and this address is to be modified by the program. Indirect Addressing may also be used for linking subroutines.

Description

Normally, the P or Q address of an instruction is the location of the data used during execution of the instruction. An indirect address, however, is the address of a second address instead of the address of data. This "second address" is the core storage address of the data to be used, that is, if the second address is not another indirect address. In effect, the address at the indirect address location is a substitute for the address of the instruction.

The data field specified by the indirect address is always five digits in length. The upper digit of the address does not require a flag bit to define the field. Moreover, its length is always five digits, even though flag bits exist within the field.

The P or Q address of an instruction is indirect when a flag bit is over the units position. Figure 23 shows that (1) the instruction (21 $0050\bar{0}$ 00650) has an indirect P address of 00500, (2) the data at 00500 is 00780, which is used as the P address during execution of the instruction, and (3) the instruction (21 $0050\bar{0}$ 00650) is not altered in core storage; only the instruction register of the 1620 is changed.

The data at the location specified by the indirect address is also an indirect address if a flag bit exists in the units position. This chaining effect continues until a flag bit does not exist in the units position of the address. The address is then treated as a direct address.

Any P or Q address of an instruction that specifies the location of data can be an indirect address. Table 1 shows the instructions that can be used in indirect addressing. When the P address of an immediate instruction is an indirect address, the Q

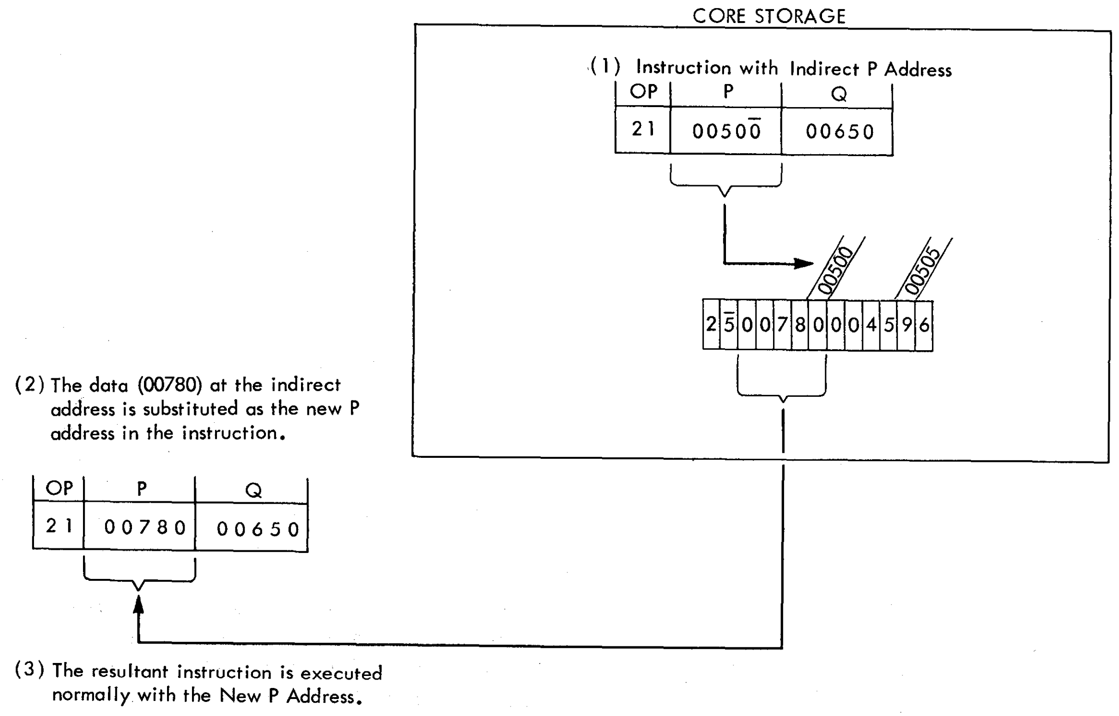

CORE STORAGE
(1) Instruction with Indirect P Address
OP P Q
21 $0050\bar{0}$ 00650
00500
00505
2 $\bar{5}$ 0 0 7 8 0 0 0 4 5 9 6
(2) The data (00780) at the indirect address is substituted as the new P address in the instruction.
OP P Q
21 00780 00650
(3) The resultant instruction is executed normally with the New P Address.

Figure 23. Indirect Addressing Data Flow

Table 1. Allowable Indirect Addressing

Instructions	Mnemonic	Code	P&Q	P
Arithmetic				
Add	A	21	X	
Add (I)	AM	11		X
Subtract	S	22	X	
Subtract (I)	SM	12		X
Multiply	M	23	X	
Multiply (I)	MM	13		X
Load Dividend*	LD	28	X	
Load Dividend (I)*	LDM	18		X
Divide*	D	29	X	
Divide (I)*	DM	19		X
Floating Add*	FADD	01	X	
Floating Subtract*	FSUB	02	X	
Floating Multiply*	FMUL	03	X	
Floating Divide*	FDIV	09	X	
Compare				
Compare	C	24	X	
Compare (I)	CM	14		X
Branch				
Branch	B	49		X
Branch No Flag	BNF	44	X	
Branch No Record Mark	BNR	45	X	
Branch No Group Mark*	BNG	55	X	
Branch on Digit	BD	43	X	
Branch Indicator	BI	46		X
Branch No Indicator	BNI	47		X
Branch and Transmit	BT	27	X	
Branch and Transmit (I)	BTM	17		X
Branch Back	BB	42		
Branch and Transmit Floating*	BTFL	07	X	
Internal Data Transmission				
Transmit Digit	TD	25	X	
Transmit Digit (I)	TDM	15		X
Transmit Field	TF	26	X	
Transmit Field (I)	TFM	16		X
Transmit Record	TR	31	X	
Transfer Numerical Strip*	TNS	72	X	
Transfer Numerical Fill*	TNF	73	X	
Floating Shift Right*	FSR	08	X	
Floating Shift Left*	FSL	05	X	
Transmit Floating*	TFL	06	X	
Input/Output				
Read Numerically	RN	36		X
Write Numerically	WN	38		X
Dump Numerically	DN	35		X
Read Alphamerically	RA	37		X
Write Alphamerically	WA	39		X
Seek*	K	34		
Program Control				
Control	K	34		
Set Flag	SF	32		X
Clear Flag	CF	33		X
Move Flag*	MF	71	X	
Halt	H	48		
No Operation	NOP	41		

* Special Feature
(I) Immediate

data cannot be more than six digits in length because the flag bit over the units position of the P address also defines the end of the immediate data.

Execution Time. Each address interpreted as an indirect address requires four additional 20-microsecond memory cycles. For example, an instruction with two indirect addresses requires an additional 160 microseconds.

Examples

The add instruction, 21 00500 00650, is shown in Figure 24 with both direct and indirect Q addresses. Line 1 shows direct addressing; the Q data is obtained from the Q address. Line 2 shows the Q address as indirect; the Q data is obtained from the address specified by the indirect address. Line 3 shows that the address specified by the indirect address is also indirect; the Q data is obtained from the address specified by the second indirect address.

The data flow diagram for an Add Immediate Instruction, 11 0050$\bar{0}$ 00650, is shown in Figure 25. The Q data $\bar{0}$00650, is added to the data at the address specified by the indirect P address. The result, $\bar{1}$155078, replaces the original P data, $\bar{1}$154428, at 09400.

A data flow diagram for a Branch instruction is shown in Figure 26. The first five digits at that indirect address are the address to which the computer branches for its next instruction.

Instructions	Data at Storage Locations			Resultant Modified Instruction	Actual Q Address Used	Actual Q Data Used
	00650	15225	12500			
①21 00500 00650	15225				00650	15225
②21 00500 0065$\bar{0}$	15225	12500		21 00500 15225	15225	12500
③21 00500 0065$\bar{0}$	1522$\bar{5}$	12500	12345	(a)21 00500 1522$\bar{5}$		
				(b)21 00500 12500	12500	12345

Figure 24. Examples of Indirect Addressing

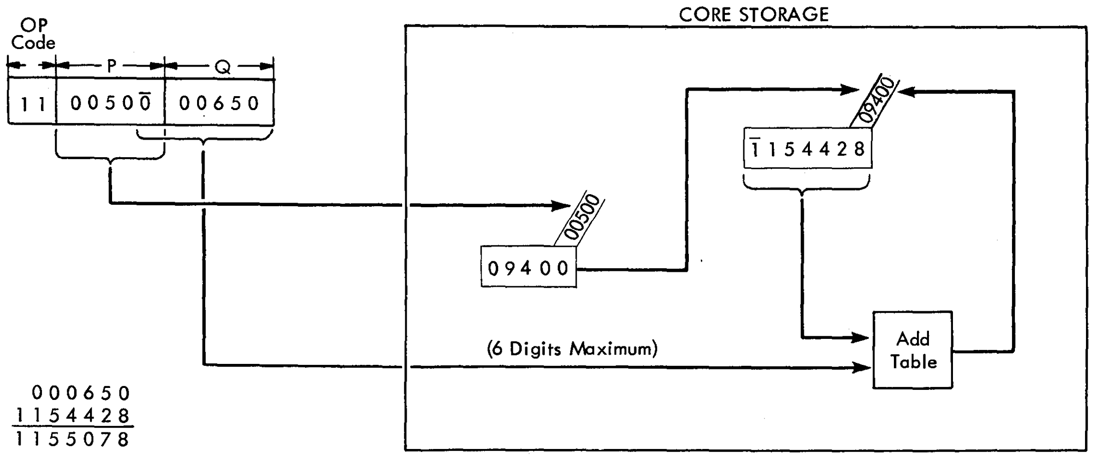


Figure 25. Indirect Addressing, Add Immediate Instruction

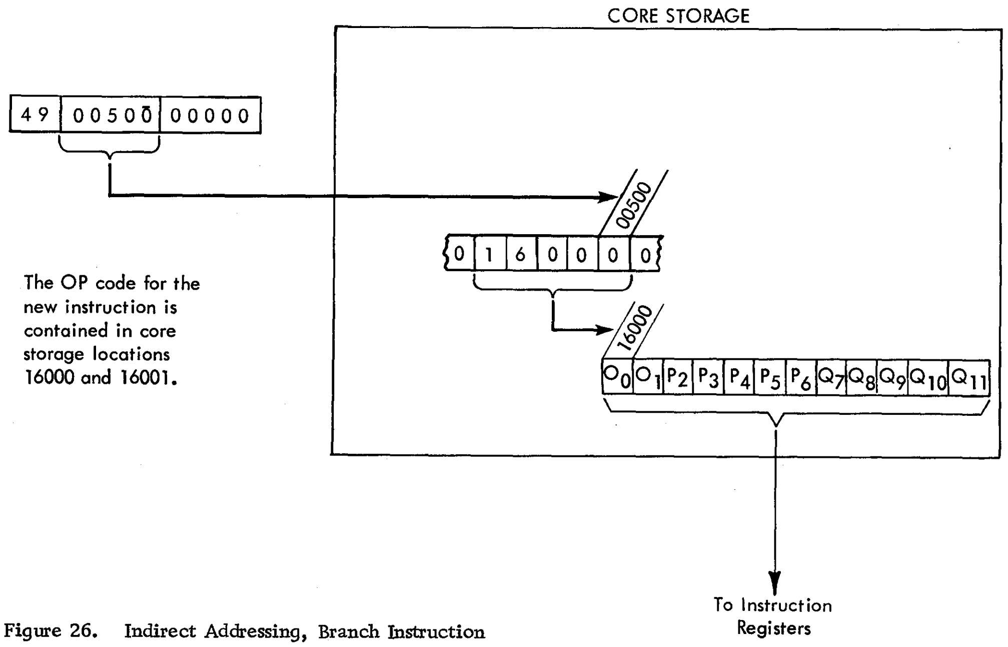

CORE STORAGE
49
0050$\bar{0}$
00000
00500
0 1 6 0 0 0 0
16000
O_0 O_1 P_2 P_3 P_4 P_5 P_6 Q_7 Q_8 Q_9 Q_{10} Q_{11}
The OP code for the new instruction is contained in core storage locations 16000 and 16001.
To Instruction Registers

Figure 26. Indirect Addressing, Branch Instruction

MOVE FLAG (MF-71)

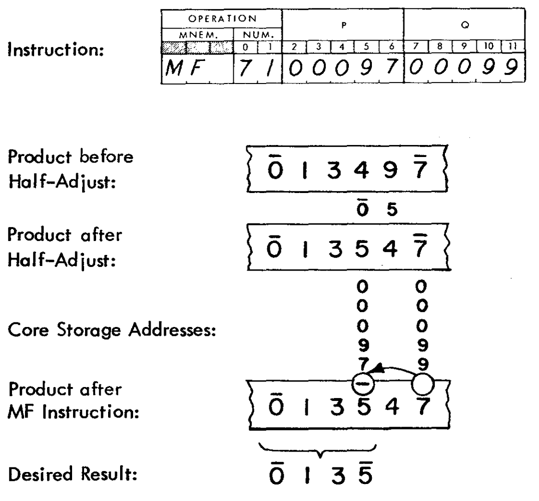
Description. This instruction moves a sign or a field definition flag from the core storage location specified by the Q address to the location specified by the P address. For example, the MF instruction moves the sign from the units position of a product to the new units position of the half-adjusted result, or moves a field definition flag to lengthen or shorten a field.

If the location specified by the Q address is without a flag, the flag at the P address is cleared. If it has a flag, that flag position is cleared and a flag is placed at the P address. Thus, after the instruction is executed, the location specified by the P address reflects the original absence or presence of a flag at the Q address, and the flag position at the Q address is clear.

Figure 27 illustrates the movement of a positive sign which, in effect, removes a flag; Figure 28, the movement of a negative sign; Figure 29, the lengthening of a field. All three illustrate the simplified programming required.

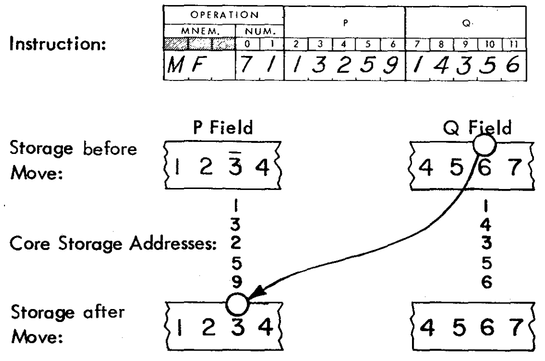
Programming is also simplified in the case where only one position of the product is dropped after half-adjustment and the product is either negative or positive. Without a Move Flag instruction, it is necessary to test for the presence of the negative flag and remove it before proper half-adjustment can take place. The simplified programming for such a situation, using the Move Flag instruction, is shown in Figure 30. The first Move Flag instruction saves the sign by placing it over its own 0_0, or high-order position. The second Move Flag instruction returns the sign to the new units position of the half-adjusted result. The flag bit can be stored in any position in which it cannot be mistaken for a field definition flag, a sign flag, or an indirect-address flag.

Execution Time. T = 240.


Instruction:
OPERATION MNEM. NUM. P Q
MF 71 00097 00099
Product before Half-Adjust: 0̄ 1 3 4 9 7̄
0̄ 5
Product after Half-Adjust: 0̄ 1 3 5 4 7̄
Core Storage Addresses: 00097 00099
Product after MF Instruction: 0̄ 1 3 5̄ 4 7
Desired Result: 0̄ 1 3 5̄

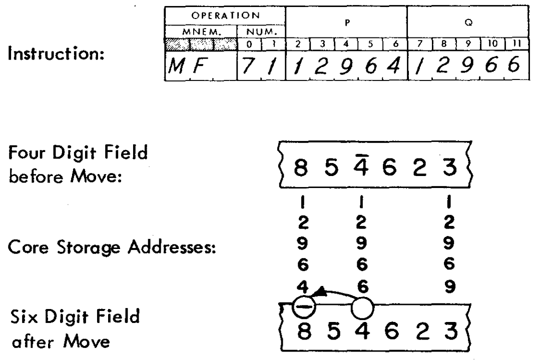
Flag is moved within same field. Note that no flag remains over the "7" after execution of the MF instruction.

Figure 28. Move Flag, Negative Sign


Instruction:
OPERATION MNEM. NUM. P Q
MF 71 13259 14356
P Field
Q Field
Storage before Move: 1 2 3̄ 4
4 5 6 7
Core Storage Addresses: 13259 14356
Storage after Move: 1 2 3 4
4 5 6 7

Flag over the "3" is cleared because there is no flag over the "6".

Figure 27. Move Flag, Positive Sign


Instruction:
OPERATION MNEM. NUM. P Q
MF 71 12964 12966
Four Digit Field before Move: 8 5 4̄ 6 2 3
Core Storage Addresses: 12964 12966 12969
Six Digit Field after Move: 8̄ 5 4 6 2 3

Flag is moved within same field. Note that no flag remains over the "4" after execution of MF instruction.

Figure 29. Move Flag, Lengthen Field

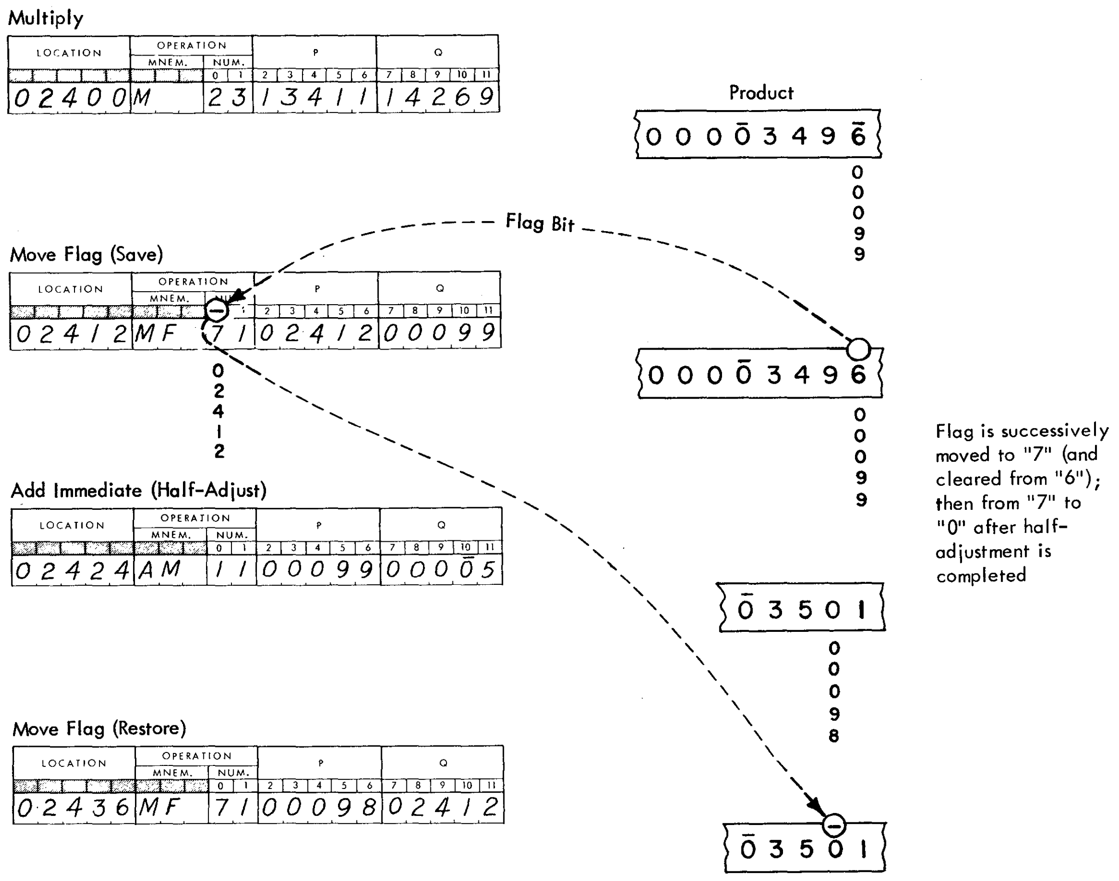

Multiply
LOCATION | OPERATION MNEM. NUM. | P | Q
02400 | M 23 | 13411 | 14269
Product
0 0 0 0̄ 3 4 9 6̄
Flag Bit
Move Flag (Save)
02412 | MF 71 | 02412 | 00099
0 0 0 0̄ 3 4 9 6
Flag is successively moved to "7" (and cleared from "6"); then from "7" to "0" after half-adjustment is completed
Add Immediate (Half-Adjust)
02424 | AM 11 | 00099 | 0000̄5
0̄ 3 5 0 1
Move Flag (Restore)
02436 | MF 71 | 00098 | 02412
0̄ 3 5 0̄ 1

Figure 30. Move Flag to Half-Adjust One Position

TRANSFER NUMERICAL STRIP (TNS-72)

Description. This instruction converts numerical data in the two-digit alphameric mode into single-digit numerical data, with sign. The units numerical position of the alphameric field is specified by the P address of the instruction which must always be an odd-numbered core storage location. The units position of the numerical field is specified by the Q address. Transmission of the numerical digits from the odd-numbered positions proceeds from the position addressed, through successively lower-numbered core storage locations, until a flag bit is sensed in other than the units position of the numerical field. The flag bit must be placed in the numerical field prior to the TNS instruction to define the high-order position. It remains unchanged by the instruction. For example, the numerical digits $\bar{4}$, 3, 2, and 1 in Figure 31 are stripped from their alphameric codes 54, 73, 72 and 71, respectively. The flag bit previously stored at 17461 terminates the operation.

The zone digits in the even-numbered core storage locations of the alphameric (P) field are ignored except for a five, two, or one in the units zone position. A five, two, or one in the units zone position is converted by TNS to a flag bit over the units digit of the numberical (Q) field. Any number other than a five, two, or one results in no flag over the units digit. A five in a units zone position of an alphamerically coded number field indicates a negative number read from an input card or

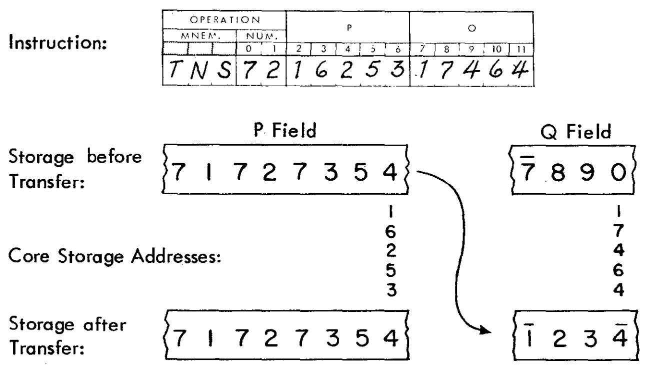


Figure 31. Transfer Numerical Strip

paper tape. A two in a units zone position occurs when an X alone, representing a negative zero, is read from an input card or paper tape. A one occurs when a negative zero (X, 0, C) is read from paper tape.

The digit in each odd-numbered core storage position of the alphameric field is transmitted without change to the corresponding position of the numerical field, concluding with the digit transmitted to the high-order position of the numerical field containing the flag that defines the field. Except for the field flag, all previous contents of the numerical field are erased by the new contents. The erasure includes any sign flag contained in the units position to designate a previous negative value. The alphameric field remains unchanged.

Flag bits in the even-numbered zone positions of the alphameric field are ignored. However, flag bits present in the odd-numbered core storage locations of the alphameric field are transmitted to the corresponding positions of the numerical field.

Because such flag bits, when transmitted, may affect the length or sign of the numerical field, all flag bit positions of the alphameric field should be cleared by instructions at the beginning of the program. Such extraneous flag bits are the result of a previous use of the core storage locations and the fact that the Read Alphamerically instruction ignores the flag bits in the read-in field. If flags are developed in the alphameric field during the program, care should be taken before the TNS instruction that the flags do not disturb the numerical field.

Execution Time. $T = 160 + 40D_p$.

TRANSFER NUMERICAL FILL (TNF-73)

Description. The TNF instruction moves and expands single-digit numerical data with sign, into two-digit alphameric data. The units numerical position of the alphameric field is specified by the P address of the instruction and must always be an odd-numbered core storage location. The units position of the numerical field is specified by the Q address.

Transmission proceeds from the location addressed, through successively lower-numbered core storage locations, until a flag bit is sensed in other than the units position of the numerical (Q) field. The digits in the numerical field, including the digit in the high-order (flagged) position, are transmitted without change to the corresponding odd-numbered positions of the alphameric field. All of the previous contents of the alphameric field, including flag bits, are erased by the new contents. The numerical field remains unchanged.

In Figure 32, the numerical digits $\bar{7}, \bar{8}, \bar{9}$ and $\bar{1}$ fill in the alphameric field locations 16251, 16253, 16255, and 16257, respectively. The field flag bit that terminates the transfer remains in the Q field and is neither transmitted nor converted.

A sign flag in the addressed units position of the numerical field is converted to a five in the even-numbered units zone position of the alphameric (P) field. Absence of a flag in the units position of the numerical field results in a seven being placed in the even-numbered units zone position. All other even-numbered zone positions of the alphameric field are automatically filled with sevens.

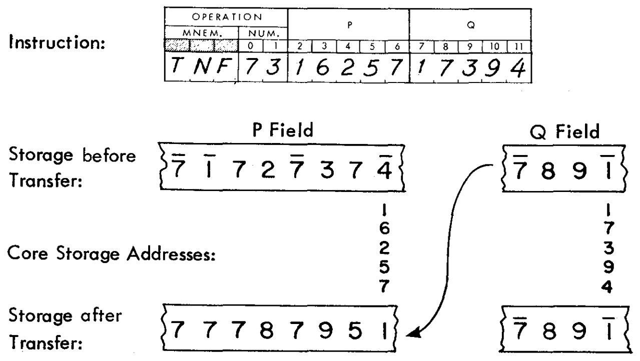


Figure 32. Transfer Numerical Fill

A26-5708-0

IBM 1620 Printed in U.S.A. A26-5708-0

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York