

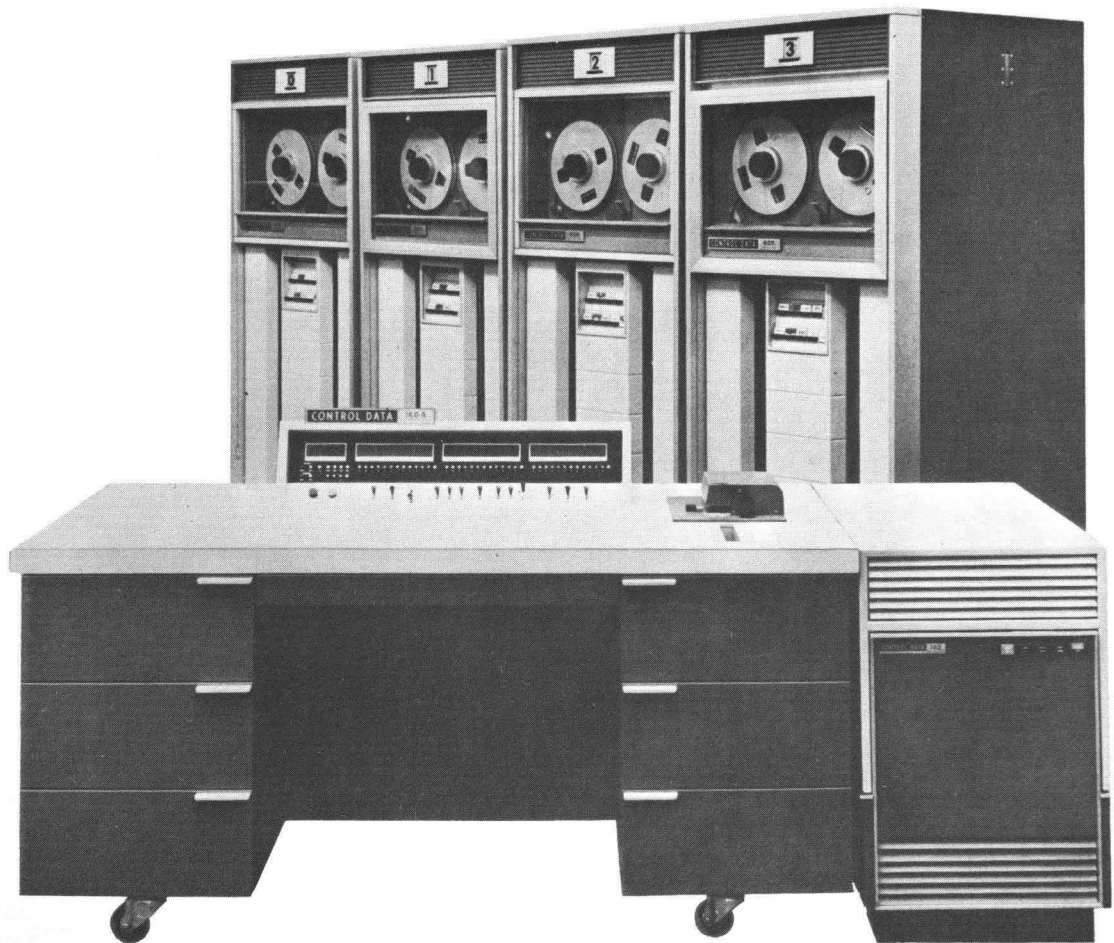
CONTROL DATA

160-A COMPUTER

160-A

160-A FORTRAN/REFERENCE MANUAL

CONTROL DATA 160-A COMPUTER



160-A FORTRAN/REFERENCE MANUAL

CONTROL DATA CORPORATION
8100 34th Avenue South
Minneapolis 20, Minnesota

This manual, publication 60051300, rev. B,
is a major revision and obsoletes publication
513A. Any comments should be addressed
to:

Control Data Corporation
Documentation Department
3145 Porter Drive
Palo Alto, California

CONTENTS

CHAPTER 1	INTRODUCTION	1
1.1	Preparation of A Fortran Program	1
1.2	Sample Program	2
CHAPTER 2	ELEMENTS OF THE LANGUAGE	5
2.1	Reserved Words	5
2.2	Constants	5
2.2.1	Integer Constants	5
2.2.2	Floating Point Constants	5
2.2.3	Masking (Boolean) Constants	6
2.3	Simple Variables	7
2.3.1	Simple Integer Variables	7
2.3.2	Simple Floating Point Variables	7
2.4	Arrays	7
2.4.1	Storage of Arrays	8
2.4.2	Subscripted Integer Variables	9
2.4.3	Subscripted Floating Point Variables	9
CHAPTER 3	ALGEBRAIC EXPRESSIONS AND STATEMENTS	11
3.1	Arithmetic Expressions, Operators	11
3.1.1	Evaluation of Exponentiation	14
3.2	Arithmetic Expressions, Mixed Mode	14
3.3	Masking Expressions	15
3.3.1	Masking Expressions, Operators	15
3.3.2	Ordering of Operations	17
3.4	Expressions as Subscripts	18
3.5	Replacement Statements	18
3.5.1	Arithmetic Statements	19
3.5.2	Masking Statements	19

CHAPTER 4	CONTROL STATEMENTS	21
4.1	Statement Identifiers	21
4.2	GO TO Statements	21
4.2.1	Direct GO TO Statement	21
4.2.2	Assigned GO TO Statement	22
4.2.3	Assign Statement	22
4.2.4	Computed GO TO Statement	22
4.3	IF Statements	23
4.3.1	Computed IF Statement	23
4.3.2	IF Sense Switch Statement	23
4.4	DO Statement	24
4.4.1	DO Loop Execution	25
4.5	Arithmetic Fault Tests	27
4.5.1	Overflow Tests	27
4.5.2	Divide Check Test	27
4.6	Continue Statement	28
4.7	Pause Statement	28
4.8	Stop Statement	28
4.9	End Statement	28
CHAPTER 5	FUNCTIONS AND SUBROUTINES	29
5.1	Functions	29
5.2	Subroutines	30
5.2.1	Subroutine Statement	31
5.2.2	Formal Parameters	32
5.2.3	Return Statement	33
5.2.4	Call Statement	33
CHAPTER 6	DATA STORAGE	37
6.1	Dimension	37
6.2	Common	38
6.2.1	Numerical Common	40
6.3	Equivalence	41

CHAPTER 7	INPUT/OUTPUT	45
7.1	Data List	46
7.2	Format	48
7.2.1	Repeating Conversion Specifications	49
7.2.2	Format Specifications	49
7.2.3	Heading and Spacing Specifications	56
7.2.4	Complete Format Specifications	57
7.3	Variable Format Control	58
7.4	Magnetic Tape Statements	59
7.5	Punched Card Statements	60
7.6	Flexowriter Statements	61
7.7	Typewriter Statements	61
7.8	Printer Statements	62
7.9	Specifications for Non-Standard Equipment	62
CHAPTER 8	CODING PROCEDURES	63
8.1	Statements	63
8.2	Continuation	64
8.3	Comments	64
8.4	Identification Field	64
8.5	Punched Cards	64
8.6	Paper Tape	65
8.7	Magnetic Tape	65
CHAPTER 9	DECK STRUCTURE	67
CHAPTER 10	SAMPLE PROGRAMS	69
APPENDIX A	UTILITY FUNCTIONS	79
APPENDIX B	DIAGNOSTICS AND MEMORY MAP	89
APPENDIX C	BCD - FLEXOWRITER - TYPEWRITER EQUIVALENCE CODES	94
APPENDIX D	GLOSSARY	97

APPENDIX E	MAGNETIC TAPE	99
APPENDIX F	PAPER TAPE AND TYPEWRITER	102
APPENDIX G	STATEMENT INDEX	104
APPENDIX H	TOPIC INDEX	107

This manual describes a FORTRAN* programming language for the CONTROL DATA® 160-A computer and is written for persons with a basic knowledge of stored-program digital computers and FORTRAN. Persons who are unfamiliar with these topics may find the FORTRAN Autotester a helpful introduction to the material in this manual.

Since most FORTRAN statements are translated into groups of machine language statements, FORTRAN programs are usually shorter, easier to write, and easier to debug than machine language programs. When the programmer's major interest is to find an immediate solution for a particular problem, FORTRAN offers him a fast and easy means to achieve this objective.

Page 2 contains a general description of the steps taken in going from a problem in the programmer's mind to the solution of the problem by the computer. Included is a sample FORTRAN problem and a description of the steps for writing this problem. In Chapters 1 - 9, the FORTRAN programming language is described; Chapter 10 contains sample programs. A glossary of computer terminology is given in Appendix D and statement and topic indexes are given in Appendixes G and H.

1.1

PREPARATION OF A FORTRAN PROGRAM

To prepare a FORTRAN program, the programmer must first reduce the solution of the problem to a series of simple steps which can be written as FORTRAN statements. The FORTRAN statements including those for transferring data into and out of the computer are written on FORTRAN coding forms and subsequently punched into cards or paper tape.

The program is read into the 160-A computer and translated by the 160-A FORTRAN compiler into an intermediate language. The intermediate language program is either executed at that time by an interpreter or stored for future use. The process of translating from FORTRAN statements to intermediate language statements is called compiling.

*FORTRAN is an abbreviation for FORMula TRANslation and was originally developed for International Business Machine equipment.

1.2

SAMPLE PROGRAM Given 20 sets of floating point constants with 4 constants in each set, find the square root of the sum of the squares of the constants in each set. If the data in each set are labeled as A(I),B(I),C(I) and D(I), where I designates the particular set (1 - 20), then the following computation is performed 20 times.

$$Y(I) = \sqrt{A(I)^2 + B(I)^2 + C(I)^2 + D(I)^2}$$

The FORTRAN program shown below reads the data from punched cards, computes Y_i for each of 20 sets and prints the answers in a column. Each line of the program contains a FORTRAN statement. The function of each statement in the program is also explained.

	<u>Statement</u>	<u>Line</u>
*	PROGRAM SAMPLE	1
	DIMENSION A(20), B(20), C(20), D(20), Y(20)	2
30	FORMAT (8F8.1)	3
	READ 30,A,B,C,D	4
	DO 4 I = 1,20	5
4	Y(I) = SQRTF(A(I)**2+B(I)**2+C(I)**2+D(I)**2)	6
40	FORMAT (10X, 1HY/(5X, F9.1))	7
	PRINT 40,Y	8
	END	9

	<u>Explanation</u>	<u>Line</u>
	The * denotes the beginning of the program. (See Deck Structure, Chapter 9.)	1
	Reserves space in computer memory for the 20 sets of constants (A,B,C,D) and 20 results (Y).	2
	Sets the input control to read floating point constants.	3
	Causes the card reader to read cards into A, B, C and D according to the format established by statement 30. One constant will be read for every 8 columns on the card; 8 constants are punched into each card.	4
	Causes the next statement to be repeated 20 times, increasing the variable I by 1 each time.	5
	Computes $\sqrt{A(I)^2+B(I)^2+C(I)^2+D(I)^2}$ and substitutes the result in the memory location reserved for Y(I).	6

<u>Explanation</u>	<u>Line</u>
Sets the output control to transmit the heading Y and a column of 20 floating point numbers.	7
Causes the printer to print the contents of Y according to the format established by statement 40.	8
Instructs the compiler that this is the end of the program.	9

2.1

RESERVED WORDS The word `FORMAT` has a special function in the FORTRAN compiler and should be used only as described in this manual. All other words including those described in this manual can be used for identifiers or variable names.

2.2

CONSTANTS

Three types of constants can be expressed in the 160-A FORTRAN language: integer (fixed-point), floating point, and masking (Boolean). The type of a constant in the source program is declared either by the form in which it is written or the environment in which it appears. All constants are stored as positive values.

2.2.1

**INTEGER
CONSTANTS**

Integer constants can have 1 to 7 decimal digits; they must be in the range 0 through 4194303. When used in an expression, a + sign or a blank before the constant indicates a positive value; a - sign indicates a negative value. If more than 7 digits are written, a diagnostic message will result. Spaces will be ignored, but illegal characters will be interpreted as the end of the constant field. (A comma is an illegal character in an integer constant). Each integer constant occupies two consecutive 160-A words.

Examples:

1	4194303
-2037	-17
+23	-6029

2.2.2

**FLOATING POINT
CONSTANTS**

A floating point constant is a number expressed as any number of decimal digits and must include a decimal point . When used in an expression, a + sign or a blank before the constant indicates a positive value; a - sign indicates a negative value. The number may be multiplied by an integral power of 10, indicated by an E and a - sign, a + sign, or a blank (indicating a +), and an integer exponent of 1 or 2 decimal digits.

The constant $.n_1n_2\dots n_mE^+e_1e_2$ is equivalent to

$$.n_1n_2\dots n_m \times 10^{e_1e_2}$$

If the E is followed immediately by a digit, the sign of the exponent is assumed to be positive.

The range of a floating-point constant is $.1 \times 10^{-32}$ through $.99999999 \times 10^{31}$. (If the CONTROL DATA® 168-2 auxiliary arithmetic unit is used, the floating point range is 10^{-38} to $10^{38}-1$). Spaces are ignored (squeezed out), but illegal characters will cause diagnostic messages.

Three consecutive machine words are used to store each floating-point constant and except for leading zeros which position the decimal point, only the first eight significant digits are retained.

Examples:

-3.9865087E22	These are equivalent forms	}	.65Eb3
.00000698E6			0.65E3
+687.830E-5			.65E+3
9.5E30			.65E03
-306.5E-30			650.0

b indicates blank

**2.2.3
MASKING
(BOOLEAN)
CONSTANTS**

A masking constant is an integer of 1 to 8 octal digits. When used in an expression, a + sign or blank before the integer indicates the value represented; a - sign before the integer indicates the seven's complement of the value represented. Masking constants represent patterns of 24 binary digits and are used with masking operators in partial word operations. Each masking constant occupies two consecutive 160-A computer words. (Each 160-A word contains 12 binary digits). Spaces and illegal characters are interpreted as zeros.

Example:

Masking Constant	Binary Representation	
	Word 1	Word 2
77000077	111111000000	000000111111
5252	000000000000	101010101010
-14	000000000000	000000001100
(when -14 occurs in an expression, the 7's complement of the stored number (111111111111 11111110011) will be used in the calculations.)		

2.3

SIMPLE VARIABLES

A simple variable is the name of a storage area in which integers, floating-point or masking constants can be stored. The variable is referenced by the location name; the value specified by the name is always the current value stored in that location.

This value can be changed at any time by an arithmetic statement or a data input statement. A simple variable has a name without a subscript and its appearance in an expression is sufficient to reserve memory space for it. Integer and floating-point variables are distinguished by the names assigned to them. Spaces are ignored in variable names.

2.3.1

SIMPLE INTEGER VARIABLES

A simple integer variable is identified by a name of from 1 to 6 alphabetic or numeric characters, the leftmost of which must be I, J, K, L, M, or N. It can be assigned any integer value in the range from -4194303 to 4194303 inclusively.

Examples:

N	NOODGE
K2S04	M58
LOX	M 58

Since spaces are ignored in variable names, M58 and M 58 are identical.

2.3.2

SIMPLE FLOATING POINT VARIABLES

A simple floating point variable is identified by a name of from 1 to 6 alphabetic or numeric characters, the first of which must be alphabetic and not I, J, K, L, M, or N.

Any value from .1 E-32 to .99999999E31 and zero can be assigned to a simple floating point variable, either positive or negative.

Examples:

VECTOR	A65302
BAGELS	BATMAN

2.4

ARRAYS

An array is a block of successive memory locations divided into areas for storage of variables. Each element of the array is referenced by the array name with a subscript. Arrays may have one, two, or three subscripts. The number of elements stored in the array is equal to the product of the subscripts.

The maximum size of a subscript or the product of the subscripts is 2047. The array name and its maximum dimensions must be declared at the beginning of the program in a DIMENSION statement. Arrays can store either integer or floating point values. The type of an array is determined by the array name; spaces are ignored in array names. A subscript can be an integer constant, an integer variable, or any integer expression listed in section 3.4.

Examples:

OBIES (9, 9) JOES (2, 8, 80)
 LOUIE (80) BILLS (40, 3, 5)

**2.4.1
 STORAGE OF
 ARRAYS**

Arrays are stored by columns as shown in the example below. Any element of an array can be referenced by the array name plus a single numeric subscript, regardless of the number of subscripts initially assigned to the array. The value of the single subscript can be determined from the following equation:

$$\begin{aligned} \text{Subscript} &= (i)+(j-1) * I+(k-1)*I*J && \text{(3 dimensions)} \\ &= (i)+(j-1)*I && \text{(2 dimensions)} \end{aligned}$$

where i, j, and k are the subscript values of the desired locations; and I, J, and K are the maximum values declared for the subscripts. The symbol * is a multiplication symbol. I and i refer to the first subscript, J and j to the second, and K and k to the third. I, J, and K in the example below equal 3.

I,J,K ARRAY (3,3,3)		
Memory Words	Array Element i, j, k	Single-Subscript Element
B+0 — B+2	B(1, 1, 1)	B(1)
B+3 — B+5	B(2, 1, 1)	B(2)
B+6 — B+8	B(3, 1, 1)	B(3)
B+9 — B+11	B(1, 2, 1)	B(4)
.	.	.
.	.	.
.	.	.
B+24 — B+26	B(3, 3, 1)	B(9)
B+27 — B+29	B(1, 1, 2)	B(10)
B+30 — B+32	B(2, 1, 2)	B(11)
B+33 — B+35	B(3, 1, 2)	B(12)
B+36 — B+38	B(1, 2, 2)	B(13)
.	.	.
.	.	.
.	.	.
B+78 — B+80	B(3, 3, 3)	B(27)

Zero as a subscript will result in a program error, an error that will not be detected during compilation, but will occur when the program is executed. Program errors will also result from the use of subscripts larger than those initially declared for the array, unless a single subscript notation is used for a two or three dimensional array. In this case, a subscript larger than the product of the declared subscripts will result in a program error.

2.4.2

SUBSCRIPTED INTEGER VARIABLES

The elements of an integer array, called subscripted integer variables, can be assigned the same values as simple integer variables. An integer array is an array named by an integer variable name (1 to 6 alphabetic or numeric characters; the first of which is I, J, K, L, M, or N).

Examples:

NEURON (6, 8, 6)	L6034(J, 3)
MORPH (20, 20)	N3(1)

2.4.3

SUBSCRIPTED FLOATING POINT VARIABLES

The elements of a floating point array, called subscripted floating point variables, can be assigned the same values as simple floating point variables. A floating point array is an array named with a floating point variable name (1 to 6 alphabetic or numeric characters, the first of which is alphabetic and not I, J, K, L, M, or N).

Examples:

TMESIS (6, 4, 7)	YCLEPT(46)
PST(20, 3, 3)	SVELTE(6, 8)

ALGEBRAIC EXPRESSIONS AND STATEMENTS

3

An expression is a constant, variable (simple or subscripted), function (Section 5.1) or any combination of these separated by operators, and parentheses, written to comply with the rules given below for expressions.

The operators are + - * / and **. Expressions are divided into arithmetic and masking expressions, and are used in arithmetic, masking, and control statements and as subscripts.

3.1

ARITHMETIC EXPRESSIONS, OPERATORS

An arithmetic expression can contain the following operators:

<u>Symbol</u>	<u>Function</u>
+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

Within an expression, sub-expressions can be grouped by parentheses to indicate the order of operations.

Two operators or two operands cannot appear next to each other in any expression. If - is used as a minus sign in an arithmetic expression, the sign and its operand must be enclosed in parentheses if it is preceded by an arithmetic operator.

Examples:

<u>Correct</u>	<u>Incorrect</u>
$-A*B+C$	$A*-B$
$B*A/(-C)$	$B*A/-C$
$A*(-C)$	$AC+B$

Hierarchy of operations:

**	exponentiation	class 1
/	division	class 2
*	multiplication	
+	addition	class 3
-	subtraction	

EXPRESSIONS WITH NO INTERNAL PARENTHESES

1. The expression is scanned from left to right until a sequence of operands and operators is found with only the operators *, /, or ** ($A^{**}B$, $A*B$).
2. If the sequence contains **, this operator is evaluated first.
3. Scanning to the left of the ** operator all * and / operations in the sequence are evaluated, left to right.
4. Then, scanning to the right of the ** operator, the remaining * and / operators in the sequence are evaluated, left to right.
5. If the sequence does not contain the ** operator, the * and / operators are evaluated as the sequence is scanned from left to right.
6. Scanning from this sequence to the right, steps 1-5 are repeated until the end of the expression is encountered. In complicated expressions, the programmer may insure the correct result by using parentheses to direct the evaluation.
7. When all sequences containing only *, /, and ** have been evaluated, or if no sequences of this type are found, the expression is scanned from left to right again and all + and - operators are evaluated as they are encountered.

Example:

$A+3*B/C-D/F^{**}2*G+H$

Order of Evaluation

$3*B$

$(3*B)/C \rightarrow T_1$

$F * * 2$

$D/(F * * 2)$

$(D/F * * 2) * G \rightarrow T2$
 $A+T1$
 $(A+T1)-T2$
 $((A+T1)-T2)+H$ evaluation complete

EXPRESSIONS CONTAINING PARENTHETICAL GROUPS

1. Expressions within parentheses are evaluated first. When parenthetical expressions contain parenthetical expressions, evaluation begins with the innermost expression and proceeds outward.
2. Each level of parenthetical expression is evaluated according to the above rules. If two or more independent parenthetical groups are on the same level, they are evaluated as they are encountered in scanning left to right.
3. After all parenthetical expressions have been evaluated, the order of operations proceeds normally until the entire expression is evaluated.

Example:

$B + A * (B + F ** 2 / (C * (D - (E + F)))) + (F - D)$

$E + F \quad \rightarrow T1$
 $D - T1 \quad \rightarrow T2$
 $C * T2 \quad \rightarrow T3$
 $F ** 2 \quad \rightarrow T4$
 $T4 / T3 \quad \rightarrow T5$
 $B + T5 \quad \rightarrow T6$
 $F - D \quad \rightarrow T7$
 $A * T6 \quad \rightarrow T8$
 $B + T8 \quad \rightarrow T9$
 $T9 + T7 \quad \rightarrow T10$ evaluation completed

Functions are treated as parenthetical groups.

The order in which operations are performed is always important, but special attention must be given to fixed point expressions where remainders of division operations are dropped.

The expression $4*10/3$ will produce a result of 13, but the expression $10/3*4$ will produce a result of 12.

3.1.1

EVALUATION OF EXPONENTIATION

The mode and final value of an expression after exponentiation can be determined from the following algorithms:

A,B = floating-point expressions

I,J = integer expressions

<u>Expressions</u>	<u>Algorithm</u>
1. A**J IF J ≥ 0 IF J < 0	A * A * . . . *A, iterated J times (1/A) * (1/A)* . . . *(1/A), iterated J times
2. A**B	$e^{B \log_e A }$
3. I**J IF J > 0 IF J = 0 IF J < 0	I*I . . . *I, iterated J times 1 0
4. I**B	$e^{B \log_e f1(I) }$ where f1(I) is the floating point value of I.

3.2

ARITHMETIC EXPRESSIONS, MIXED MODE

Integer and floating-point variables and constants may appear in the same arithmetic expression. The following are legal 160-A FORTRAN expressions

JOE + BETA(31)*KATT/R(6)

ADAM/L(2,3) + IND + BROOM/6

Mixed mode expressions are evaluated according to the rules in Section 3.1 with the modification that whenever an operation involves both an integer and a floating-point operand, the integer is converted to a floating-point quantity before the operation is performed. Computing time can be shortened by arranging arithmetic expressions to optimize the number of integer-to-floating-point conversions.

The expression $I * ABE/J$ requires two conversions whereas the equivalent expression $(I/J)*ABE$ requires only one. (Because of truncation during integer division, however, these two expressions may produce different values.)

3.3

MASKING EXPRESSIONS

Masking expressions are used primarily to mask selected parts of constants and variables for arithmetic operations. With masking expressions, it is possible to remove any particular bits of an operand, add them to the same bits of another operand, and restore the results in the first operand.

All expressions except subscript expressions are interpreted as masking expressions if a B appears in column 1 of the coding form. Subscript expressions are always arithmetic even when they occur in a masking expression.

Example:

B A(I,J) = C(3*K+2) + R(5*I,3*J-2) the expression is masking
and contains arithmetic
subscript expressions

The following rules apply to masking expressions:

1. Operations are performed on a bit-by-bit basis with 24-bit operands.
2. If a floating-point quantity is specified, only the first 24 bits will be used.
3. Expressions can be connected by operators to form more complex expressions with the restriction that the minus sign specifies an operation on the operand immediately to the right only. For example, A - B has no meaning because the minus sign specifies an operation on B only. One correct expression would be A+(-B).

3.3.1

MASKING EXPRESSIONS, OPERATORS

A masking expression can contain the following operators:

<u>Symbol</u>	<u>Function</u>
-	Complement (NOT)
+	Inclusive OR
/	Exclusive OR
*	AND
**	Shift

The NOT operator specifies the bit-by-bit complement of the operand immediately to the right. (All zeros are changed to ones and all ones changed to zeros.)

	<u>Constants</u>	<u>Complement</u>
Octal	77000700	00777077
	75400023	02377754
Binary	110111000101011100111000	001000111010100011000111

The inclusive OR, exclusive OR, and AND operators are defined by the following tables, where the abscissa is a bit position of one operand while the ordinate is the corresponding bit position of the other operand.

+	0	1	/	0	1	*	0	1
0	0	1	0	0	1	0	0	0
1	1	1	1	1	0	1	0	1

	<u>Binary</u>	<u>Octal</u>
Operands	A=101111010011001100000111	57531407
	B=101001001011110000000001	51136001

<u>Expressions</u>	<u>Results</u>	
A+B	101111010011111100000111	57537407
A/B	000110100000111100000110	06407406
A*B	101001001011000000000001	51130001

The operator ** is defined for the expression I**J as follows:

- J ≥ 0 I is shifted J binary positions to the left. Bits leaving the pattern on the left reappear on the right (end around to the left).
- J < 0 I is shifted |J| binary positions to the right. Bits leaving the pattern on the right are lost (end off to the right). Sign is not extended.

J must be either an octal number or an integer variable.

<u>Expression</u>	<u>Result</u>
00774073**6	77407300
00774073**7	77016601
00774073**(-2)	00177016

(all numbers are octal numbers)

3.3.2 ORDERING OF OPERATIONS

The ordering of operations in masking expressions is as follows:

EXPRESSIONS WITH NO INTERNAL PARENTHESES

1. The expression is scanned from left to right until a sequence of operands and operators is found with only the operators -, **, or *.
2. If the group contains -, this operator is evaluated first.
3. Scanning left, all ** and * operators in this group are evaluated.
4. Scanning right, the remaining ** and * operators in the group are evaluated.
5. If the group does not contain the - operator, the ** and * operators are evaluated as the group is scanned from left to right.
6. Scanning from this group to the right, steps 1-5 are repeated until the end of the expression is encountered.
7. When all groups containing only -, **, and * have been evaluated, or if no groups of this type are found, the expression is scanned from left to right again and the + and / operators are evaluated as they are encountered.

Example:

-I + J + K*L**7

-I → R1

K*L

(K*L)**7 → R2

R1 + J → R3

R3 + R2 → R4

EXPRESSIONS CONTAINING PARENTHETICAL GROUPS

The innermost parenthetical group is evaluated according to the above rules. Two or more parenthetical groups on the same level are evaluated as they are encountered in scanning left to right. After all parenthetical expressions have been evaluated, the order of operations proceeds according to rules 1 to 7 until the entire expression is evaluated.

Example:

$J+(-I) + K*(L^{**7} + M)$

$-I$

L^{**7}

$(L^{**7})+M$

$K* ((L^{**7})+M) \rightarrow T1$

$J+(-I) \rightarrow T2$

$T2 + T1 \rightarrow T3$

3.4

EXPRESSIONS AS SUBSCRIPTS

The following integer expressions can be used as subscripts for array elements.

n	$i-n$
i	$m * i+n$
$m*i$	$m * i-n$
$i+n$	

m and n are unsigned integer constants and i is a simple integer variable. A subscript expression is always evaluated as an arithmetic expression, even if it appears in a masking statement.

$CLOTHO(MOERAE-3)$

$ATROPO(12*KUTS)$

$FATAE(3*NEMES+80)$

3.5

REPLACEMENT STATEMENTS

Replacement statements may be arithmetic or masking in the form $R = E$. The symbol $=$ is a replacement operator which specifies that the expression E is to be evaluated and the result stored in R .

3.5.1

ARITHMETIC STATEMENTS

In the statement $R = E$, R is any variable name (simple or subscripted) and E is any arithmetic expression. R will be replaced by the value of the expression E with mode change, if necessary.

The mode of the variable name R determines the mode to which the value of the expression will be converted. For example, in the statement $A = I + 2$, the value of $I + 2$ will be calculated in integer mode and converted to floating point mode before being stored in A . In the statement $I = X + 2.0$, the value of $X + 2.0$ will be calculated in floating point mode and converted to integer mode by dropping the fractional part.

If, in the statement $IP = F + Y$, the evaluation of the expression $F + Y$ resulted in a value of 2.56, IP would be assigned the value 2.

```
UB(1)    = (X+Z)*Y/60.3*(X**2)
IST      = IST + 1
IGNIS    = EZRA (K)+B (3)
```

3.5.2

MASKING STATEMENTS

A masking statement has the same form as an arithmetic statement and is identified as masking by the letter B in column 1 on the coding form. In the statement $R = E$, R is a simple or subscripted variable name and E is a masking expression. The operator $=$ in a masking statement means replace by.

Integer and floating point constants in masking expressions are evaluated as octal numbers; operations are performed on a bit-by-bit basis with 24-bit operands. Masking expressions can be assigned to either integer or floating point variables. If a masking operation is specified on a floating point quantity, only the first two of the three computer words for that quantity are used.

```
A        = X+(-B)
JOE(3)   = (X*Y)+(Z)*7700
```

The execution of statements normally proceeds from one statement to the statement immediately following it in the program. Control statements can be used to alter this sequence and transfer control to any other statement in the program, or they can cause a number of iterations of a program section. Control may be transferred only to executable statements (Appendix G). A transfer to a non-executable statement will result in a program error, but not a compiling error.

Iteration control provided by the DO statement causes a predetermined sequence of instructions to be repeated any number of times with the stepping of a simple integer variable after each iteration. PAUSE, STOP, and END statements provide for termination of the main program or subroutines.

4.1 STATEMENT IDENTIFIERS

Statements are identified by numbers which can be referenced in other sections of the program. On the coding form, statement identifiers appear on the same line as the statement. Statements containing masking expressions have 1 to 4 digit identifiers in columns 2 through 5 with the letter B in column 1. All other statement identifiers are from 1 to 5 digits and are written in columns 1 through 5. Leading blanks and zeros are ignored; the following identifiers are equivalent forms:

05
5
0005

4.2 GO TO STATEMENTS

Unconditional transfer of control is provided by GO TO statements which transfer control either to a fixed address or an address which is determined during execution of the program.

4.2.1 DIRECT GO TO STATEMENT

GO TO n

Control will be transferred to statement n; n is a statement identifier. Since the value of n cannot be altered, this statement is used for unconditional transfer of control to a fixed location.

```

GO TO 20
.
.
.
20  SUM = SUM + 1

```

**4.2.2
ASSIGNED GO
TO STATEMENT**

GO TO i

Control will be transferred to the current statement number assigned to the simple integer variable i by an ASSIGN statement. The variable i represents a statement label; the variable name can be used in arithmetic expressions. The assigned GO TO is used to alter the transfer address during the execution of the program.

```

GO TO JERRY
GO TO M

```

**4.2.3
ASSIGN
STATEMENT**

ASSIGN n TO i

This statement assigns an identifier to the simple integer variable i; it is used only in conjunction with the assigned GO TO statement.

```

30  KOUNT = A*B
    ASSIGN 30 TO JERRY
    .
    .
    .
    GO TO JERRY
    .
    .
    .
    ASSIGN 40 TO JERRY
    .
    .
    .
40  KOUNT = A/B

```

**4.2.4
COMPUTED GO
TO STATEMENT**

GO TO (n₁, n₂, . . . , n_m)i

GO TO (n₁, n₂, . . . , n_m),i

n₁, n₂, . . . , n_m are statement numbers, one of which will be executed next and i is a simple integer variable which may assume one of the subscript

values in the sequence 1, 2, . . . ,m. Each time this statement is encountered, the current value of i determines the transfer address. The value of i is assigned by an arithmetic statement in the program and it can be altered as often as the programmer desires.

The computed GO TO is used when transfer of control is contingent upon the current computed value of an integer variable. A program error will result if i is greater than the number of addresses.

```

        I = 2
        GO TO (80,33,9,70),I           control is transferred to statement 33
        .
        .
        .
33     J = I + 1
        GO TO (10,11,12)J           control is transferred to statement 12

```

4.3

IF STATEMENTS

Conditional transfer of control is provided by IF statements.

4.3.1

COMPUTED IF STATEMENT

IF (E) n_1, n_2, n_3

E is an arithmetic expression (or masking expression if a B appears in column 1 of the coding form) and n_1, n_2, n_3 are statement identifiers. This statement provides a three way branch in the program contingent on the value of an expression. If E is less than zero, n_1 will be executed. If E is zero, n_2 will be executed; if E is greater than zero, n_3 will be executed.

```

        IF (I*JOE(3)/N(7,4) )10,11,10
B     IF (X+(-Y) ) 20,21,22

```

4.3.2

IF SENSE SWITCH STATEMENT

IF (SENSE SWITCH i) n_1, n_2

Statement n_1 will be executed if sense switch i is on; if it is off, statement n_2 will be executed. i is an integer 1-7 and n_1, n_2 are statement numbers. The three sense switches on the 160-A console are referenced by digits 1, 2, and 4; they are normally off and must be preset by the operator before the program is run. The inclusive OR function of sense switches can also be used. The sense switch test may be used with the dump functions (Appendix A).

This statement is generally used during debugging to control the sequence of the program from the console.

The following combination of switches are tested:

<u>i</u>	<u>sense switches</u>
3	1,2
5	1,4
6	2,4
7	1,2,4

Thus, if i is 3, n_1 will be executed if either sense switch 1 or 2 is on.

4.4

DO STATEMENT

DO n i = m_1, m_2, m_3

The group of statements which follow, up to and including statement n , will be repeated the number of times specified by the values m_1, m_2 and m_3 . The index, i , is a simple integer variable; and m_1, m_2 , and m_3 are unsigned integer constants or simple integer variables. Positive or negative values can be assigned to integer variables used in DO statements. The initial value assigned to i is m_1 , m_2 is the largest value assigned to i , and m_3 is the amount added to i after each DO loop is executed. If m_3 is one, it can be omitted. The statements starting with the DO and ending with n form the DO loop. The range of the DO loop is the DO loop excluding the DO statement. In the program

```
      .  
      .  
      .  
100 DO 101 I = 1, 10, 2  
101 A(I) = B(I + 2)
```

the range of the DO statement is statement 101 and the DO loop consists of statements 100 and 101.

Examples:

```
DO 6 I = 5,10,2  
DO 2 K = -30,30  
INCR = -1  
DO 5 J = 30,0,INCR
```

4.4.1

DO LOOP EXECUTION

The DO loop is executed in the following manner:

1. Index i is set to m_1 .
2. The expression $m_4 = (m_1 - m_2) / m_3$ is computed and the integer part retained.
3. The loop is executed once.
4. If m_4 is positive (zero is positive), the loop is satisfied and the next statement following the DO loop is executed.
5. If m_4 is negative, one is added to m_4 .
6. m_3 is added to i .
7. Steps 3-6 are repeated.

Example :

```
      .
      .
      .
      DO 10 I = 2,8,2           Index i becomes 2,4,6,8 successively.
      AJ = J                    $m_4 = (2-8)/2$ 
10  PROD = PROD * AJ           $m_4 = -3$ , the loop will be executed 4 times
11  SUM = PROD + SUM          before finally satisfied.
      .
      .
      .
      .
```

Control will then be transferred to statement 11.

Any statement may be included in the range of a DO statement with the following restrictions.

1. The last statement in the range of a DO loop may not be a transfer of control statement (GO TO, IF (E)). See CONTINUE (Sec. 4.6).
2. A DO statement in the range of another DO statement must have a range which is contained in the first DO statement range; that is, the last statement of the inner DO loop must either be the same statement as the last statement of the outer DO loop, or occur before it.

These statements are correct:

```
.  
.   
.   
DO 10 I = 1, 10  
.   
.   
DO 11 J = 2, 20  
.   
.   
11 --  
.   
.   
10 --  
.   
.   
.
```

The statements below are incorrect because the inner DO loop is not wholly contained in the outer DO loop range:

```
.  
.   
.   
DO 10 I = 1, 10  
.   
.   
DO 11 J = 2, 20  
.   
.   
10 --  
.   
.   
11 --
```

3. The number of repetitions of the DO loop is determined when the loop is entered and cannot be altered after this point. The value of the loop index may be altered within the loop by the programmer, but this will not affect the number of repetitions of the loop.
4. The variable used as the index of a DO statement may be used as an integer variable within the DO loop.

5. Once the DO loop has been initiated, a program can:
 - a. transfer out of the range of a DO loop,
 - b. transfer out then back into the same DO loop,
 - c. transfer from an inner DO loop into a more inclusive loop in the case of nesting (rule 2).

4.5

ARITHMETIC FAULT TESTS

The results of floating point operations on the standard 160-A computer must remain in the range $.1 \times 10^{-32}$ through $.99999999 \times 10^{31}$. If the CONTROL DATA[®] 168-2 auxiliary arithmetic unit is used, however, the floating point range is 10^{-38} to 10^{38} . If a floating point operation produces a result outside of this range, an overflow switch is turned on. Two statements may be used to test for overflow faults. The actual result of such an operation will be an un-normalized floating-point constant with exponent E32.

4.5.1

OVERFLOW TESTS

IF ACCUMULATOR OVERFLOW n_1, n_2
IF QUOTIENT OVERFLOW n_1, n_2

If the overflow switch is on, control is transferred to statement number n_1 and the overflow switch is turned off. If the overflow switch is off, a transfer is made to n_2 .

This statement should be used to test for overflow whenever floating point operations are performed which may produce results outside of the 160-A floating point range.

4.5.2

DIVIDE CHECK TEST

IF DIVIDE CHECK n_1, n_2

If the divide check switch is on, control is transferred to statement n_1 and the divide check switch is turned off. If the divide check switch is off, a transfer is made to n_2 . This statement is used to detect an attempt to divide by zero.

4.6

CONTINUE STATEMENT

CONTINUE

CONTINUE is a dummy statement primarily used as the last statement in a DO loop range when the last statement would otherwise have been a transfer statement. It may be used to establish a label at any point in a program. During program execution, it is interpreted as a no operation statement, and control continues to the next statement. It must not be used as the first statement in a subroutine.

4.7

PAUSE STATEMENT PAUSE n

PAUSE n stops the computer with the last two digits of n displayed in the least significant digits of the accumulator; n is an octal integer or blank. The upper two digits of the accumulator will be 00. If the run switch is pressed, program execution will continue with the statement following the PAUSE statement. This statement is used primarily in debugging a program.

PAUSE (n omitted) causes a pause in computation with zeros displayed in the accumulator.

4.8

STOP STATEMENT STOP n

STOP n stops the computer with the last two digits of n displayed in the least significant digits of the accumulator; n is an octal integer or blank. The upper two digits of the accumulator will be 00. Computation cannot be continued past this point.

STOP (n omitted) causes a stop with all zeros in the accumulator.

4.9

END STATEMENT END

The last statement of the main program and of each subroutine must be an END statement. In a main program this statement automatically forms an additional statement which is STOP 00. The STOP 00 statement effectively precedes the END statement.

Functions and subroutines are logically independent statements or groups of statements which can be called at any time for execution. A function is a group of statements which computes a value like sin, cos, or log. Functions are stored on the 160-A FORTRAN library tape and are called for execution whenever their names appear in an arithmetic expression. A function can operate on any number of variables or constants, but it can return only a single result. The values which the function operates on must be communicated directly to the function when it is called.

A subroutine is an independent program that must be called for execution by a CALL statement. It can operate on any number of variables or constants and can return any number of results. Subroutines can communicate with other sections of the program either through explicit transmission of values or through COMMON.

5.1 FUNCTIONS

The standard 160-A FORTRAN library contains eight functions. Additional functions can be included at the option of the installation or standard functions can be expanded or replaced. In addition, a number of utility functions are available. See Appendix A. Only those routines required from the library are compiled with a particular program. A function name consists of 1 to 6 alphanumeric characters ending in F, with the parameters in parentheses. The value of a function is an integer if the first character of the function name is X.

<u>Standard Library Functions</u>	<u>Computation</u>
SINF(X)	Sine of X radians
COSF(X)	Cosine of X radians
ATANF(X)	Inverse tangent in radians ($-\pi/2$ to $\pi/2$) of X
EXPF(X)	e raised to the power X
LOGF(X)	Natural logarithm of X
† ABSF(X), XABSF(I)	Absolute value of X or I

† These functions are built into the compiler, but they are used as if they were library functions.

<u>Standard Library Functions</u>	<u>Computation</u>
SIGNF(X,Y)	Sets the value of X with the sign of Y
SQRTF(X)	Square root of X

The values in parentheses following the function name (X,Y, and I) are referred to as formal parameters.

A function is used as an operand in an arithmetic expression. When the function is called by appearing in an arithmetic expression, the actual parameter values are enclosed in parentheses after the function name in the same order as the formal parameters to which they correspond. Thus, in the following expression

$$B(I) * \text{COSF}(\text{POE}(8,3)) / 38$$

the function COSF would be computed with X=POE(8,3) and the result substituted for the operand COSF(POE(8,3)). The remainder of the expression would then be computed.

The actual values used in the function call can be constants, variables or expressions; but they must correspond in order, mode (integer or floating point) and number with the formal parameters. The correspondence of actual and formal parameters is always by position; that is, the first parameter after the left parenthesis in the function call is substituted for the first formal parameter after the left parenthesis in the function definition, and so on.

Examples:

$$Z = \text{SINF}(\text{BETA})$$

$$W = \text{COSF}(-23 ** SK)$$

$$\text{ALPHA} = \text{SIGNF}(I * A, -J * B)$$

$$\text{OMEGA} = \text{SQRTF}(\text{SINF}(\text{ATANF}(X)) / \text{COSF}(\text{ATANF}(Y)))$$

**5.2
SUBROUTINES**

A subroutine is an independent program that can be called for execution by the main program or by another subroutine. It is generally used when the same set of instructions must be executed at different points in a program.

A subroutine can operate on information present in the main program and can return a value or values resulting from these operations to the main program. This communication of information can be accomplished in two ways.

1. A program can store information in the common area of storage in locations which are referenced by the subroutine. This is referred to as implicit transfer of information. (section 6.2)
2. A program can also transmit information through the formal parameters of the subroutine and the subroutine may return results in a similar fashion. This is referred to as explicit transfer of information. The formal parameters for subroutines and functions transmit information in the same way.

Although subroutines must be compiled with the main program, they are independent of the main program and of other subroutines. This means that variable names and statement identifiers that appear in one program section (the main program or a subroutine) are not defined outside of that section and can not be referenced directly by any other program section.

5.2.1

SUBROUTINE STATEMENT

SUBROUTINE name
or
SUBROUTINE name (a_1, a_2, a_3, \dots)

Name is an identifier of 1 to 6 alphabetic or numeric characters, the first of which must be alphabetic. The items a_1, a_2, a_3, \dots are the formal parameters of the subroutine.

The complete subroutine begins with a SUBROUTINE statement, includes any legal 160-A FORTRAN statements except another SUBROUTINE statement and is terminated by an END statement. The following rules apply to subroutines:

1. If a COMMON statement is used, it follows the SUBROUTINE statement.
2. If a DIMENSION statement is used with a COMMON statement, DIMENSION appears immediately after COMMON.
3. If a DIMENSION statement is used and a COMMON statement is not used, the DIMENSION statement follows the SUBROUTINE statement.
4. A subroutine may call other subroutines and may also use functions.
5. The first statement after the SUBROUTINE statement must not be a CONTINUE statement.
6. Subroutines may appear before the main program, but must not appear between statements in the main program or within another subroutine.
7. A subroutine may have a maximum of 15 formal parameters, or it may have none.

5.2.2 FORMAL PARAMETERS

The following rules apply to formal parameters:

1. A formal parameter can be a simple variable name or the name of an array. All restrictions on size, name, and mode of simple variables and arrays apply also when they are used as formal parameters.
2. The actual parameters supplied when the subroutine is called must agree with the formal parameters in order and mode.
3. If a formal parameter references an array, the array name with its critical subscripts must appear in a DIMENSION statement in the same subroutine. The critical subscript for a two dimensional array is the first subscript; for a three dimensioned array, the first two subscripts. The array names which will be actual parameters must appear in a DIMENSION statement in the main program. For example, the subroutine

```
SUBROUTINE BARB(X,Y)
  DIMENSION X(10,10), Y(10,10)
  .
  .
  .
```

uses the formal parameter X as an array which will have maximum subscript values of 10,10. The main program DIMENSION statement would appear as

```
DIMENSION A(10,10),B(10,10)
```

and a call for the function BARB would appear

```
CALL BARB(A,B).
```

4. A subroutine can have none, one, or up to 15 formal parameters and can return none, one, or more results.
5. The values of formal parameters having been determined through arithmetic expressions or data input, are substituted for the actual parameters to which they correspond after the subroutine is executed.
6. Because formal parameter names are local to the subroutine in which they appear, they may be the same as names appearing outside the subroutine.

7. A formal parameter can be used both as an input and as an output parameter. In the subroutine shown below, JOBX will be assigned the value of the corresponding actual parameter before the subroutine is executed. The subroutine will then be executed, using this value for JOBX. After execution the new value of JOBX will be substituted in the corresponding actual parameter (I). I will assume the new value for the remainder of the program unless changed.

Subroutine

```

SUBROUTINE COUNT (JOBX, JOBY)
  JHOURS = JOBX*36+JOBY/6*JOBX
      .
      .
      .
  JOBX = (JHOURS**12)/48
  END

```

Calling Program Reference

```

      .
      .
  CALL COUNT (I,J)
      .
      .
      .

```

5.2.3

RETURN STATEMENT

RETURN

This statement returns control from a subroutine to the statement immediately following the CALL statement. (If the CALL statement ends a DO loop, the loop will be continued until it is satisfied.)

Every subroutine must be terminated by an END statement. This statement returns control to the statement following the CALL statement if a RETURN statement is not used.

5.2.4

CALL STATEMENT

CALL name (a_1, a_2, \dots)

Control transfers to the subroutine name for execution; (a_1, a_2, \dots) are the actual parameters to be used in the subroutine.

Each of the actual parameters of the CALL statement must be one of the following forms (integer and floating-point):

constant
simple or subscripted variable
array name
arithmetic expression

The actual parameters in the CALL statement must agree in number, mode, order, and with the formal parameters in the SUBROUTINE statement of the called subroutine.

EXAMPLES OF SUBROUTINES

- (1) SUBROUTINE AA(BCD,PO,DIV)
BCD=PO-(PO/DIV)*10
IF(BCD)2,3,2
2 RETURN
3 BCD=10
END
SUBROUTINE DIVINE (WATER,RODS,HUNJUN)
HUNJUN=10
CALL AA(VOODOO,WATER,HUNJUN)
PRINT 10, VODOO
10 FORMAT (1X,A1)
END
- (2) Example of subroutine within a main program. Note in particular the use of COMMON.
- * PROGRAM MEASURE
COMMON DEBIT
DIMENSION DEBIT (1000), BAL (1000)
CALL MEAN(AVER)
DO 5 I = 1,500
5 BAL (I) = DEBIT(I)+AVER
DO 10 I = 501,1000
10 BAL(I) = DEBIT(I)-AVER

```
END
SUBROUTINE MEAN (AVER)
COMMON OWE
DIMENSION OWE (1000)
CTR = 0.
AVER = 0.
DO 15 I = 1,1000
IF(OWE(I)-50.) 6, 15, 15
6  CTR = CTR + 1
   AVER = AVER + OWE
15 CONTINUE
   AVER = AVER/CTR
END
```


EXAMPLES OF PROGRAMS WITH SUBROUTINES

<u>Program</u>	<u>Operations Performed</u>
<pre> * PROGRAM 1256 DIMENSION A(20) . . XLOBT=A(6)*65.8/A(9)**3 . . CALL PUTTY(A,SQRTF(XLOBT),ISAQ) . . SUBROUTINE PUTTY(WYE,XINK,LYNK) DIMENSION WYE(20) WYE(4)=WYE(1)**2+WYE(3)**2 LYNK=WYE(4)/XINK+LYNK*36 END </pre>	<ol style="list-style-type: none"> 1. Starting address of array A is substituted for address of WYE. 2. Function SQRTF is evaluated using current value of XLOBT, and result is used for XINK. 3. Contents of ISAQ are used for LYNK. 4. Subroutine is executed. 5. Contents of LYNK are stored in ISAQ. 6. Control is returned to statement following CALL PUTTY ().
<pre> * PROGRAM REPORT DIMENSION NEWS(10,10) . . . CALL DATA(NEWS) . . SUBROUTINE DATA(MRAVDA) DIMENSION MRAVDA(10,10) READ 10, MRAVDA 10 FORMAT (A4) END </pre>	<ol style="list-style-type: none"> 1. Starting address of NEWS is substituted for address of MRAVDA. 2. Subroutine is executed, reading values into NEWS(1,1) - NEWS(10,10). 3. Control is returned to statement following CALL DATA ().

DIMENSION and COMMON statements allocate storage space for variables used in the program; the EQUIVALENCE statement permits variables to share storage space. All three statements are non-executable and must appear before the first executable statement (Appendix G) of the program. When all the statements are used in a program, the order of sequence is:

```
COMMON
DIMENSION
EQUIVALENCE
```

6.1

DIMENSION

```
DIMENSION V1, V2, V3, . . . .
```

where V_1, V_2, V_3, \dots are subscripted variable names having 1, 2, or 3 integer constant subscripts (s_1, s_2, s_3).

Memory locations will be reserved for the arrays V_1, V_2, V_3, \dots . The subscripts s_1, s_2, s_3 of the variable name determine the number of memory locations required. The number of locations in the array will be equal to the product $s_1 * s_2 * s_3$ times the number of words for each variable (two for integer, three for floating point). Each subscript for an array states the maximum value which it may be expected to assume during execution of the program.

Every subscripted variable in a program or subroutine must appear with its subscripts in a DIMENSION statement in that program or subroutine.

A single DIMENSION statement may contain any number of array names, and any number of DIMENSION statements can be included in a program or subroutine.

Example:

```
DIMENSION A(3,9,6),JOE(55),ZEKE(8,9)
```

DIMENSION statements that dimension arrays contained in common must immediately follow the COMMON statement.

6.2

COMMON

Variable names are not defined outside of the program or subroutine in which they appear. In the following program LOGOS, UPS, PCOUNT, and RATIO are defined for the main program only and INN, OWT, MIN, MAX, and PCOUNT are defined for the subroutine only. The variable PCOUNT defined in the subroutine is not assigned to the same location as the variable PCOUNT in the main program, and therefore is independent of it.

```
DIMENSION LOGOS(12,12), UPS(3,3,3)
.
.
.
PCOUNT = LOGOS(3,6)*RATIO/12
.
.
.
SUBROUTINE SEARCH(INN, OWT)
.
.
.
PCOUNT = OWT*INN + OWT/INN
.
.
.
OWT = INN + (MAX-MIN)/2 + PCOUNT
.
.
.
END
```

To transmit information from one program section (main program, subroutines) to another, without using formal parameters of subroutines, the information must be stored in an area called common. Information in this area is available to the main program and to all subroutines. Storage locations are reserved in common by the statement

```
COMMON L1, L2, L3
```

L_1, L_2, L_3 are simple variable names or array names without subscripts.

Information is exchanged between program sections by having variables in the different sections occupy the same areas in common. To insure the transfer of information, a COMMON statement must appear in each program section which is to share information. Variable names used in the program sections are defined only for the section in which they appear, even if they appear in common.

Example:

```
COMMON AA, JOB, NOS
DIMENSION AA(20)
.
.
.
SUBROUTINE SALARY(PX, RYTE)
COMMON EMP, LYMT, NET
DIMENSION EMP(20)
.
.
.
```

In this example, the array AA in the main program will occupy the same area in common as the array EMP in subroutine SALARY. JOB will occupy the same area as LYMT and NOS will occupy the same area as NET.

CAUTIONS

(1) Since integer variables occupy two memory words and floating point variables occupy three memory words, it is necessary that the variable names agree not only in position within the COMMON statement, but also in mode. If, in the previous example, JOB were replaced by BOB, it would occupy one more word in common than LYMT and NOS and NET would no longer reference the same location.

(2) Variables are stored backwards in common. Thus, the statements

```
COMMON LAKE, FISH, WEEDS
DIMENSION LAKE(2,2)
```

will result in the following assignments within common:

Location (memory words) relative to WEEDS	Variable
1-3	WEEDS
4-6	FISH
7-8	LAKE(1,1)
9-10	LAKE(1,2)
11-12	LAKE(2,1)
13-14	LAKE(2,2)

(3) If an array appears in a COMMON statement, the array must also appear in a DIMENSION statement which immediately follows the COMMON statement.

(4) If a program section does not use all of the common locations reserved by other program sections, it may be necessary to include dummy variables (variables not used in that program section) in the COMMON statement in that section to ensure the proper correspondence of common areas. Thus, if a subroutine were to use the information stored in LAKE(2,2) and WEEDS in the above example, the statements

```
COMMON I,A,B  
DIMENSION I(2,2)
```

could be used. The variable A is a dummy variable used to space over the area reserved by FISH. If only the information stored in LAKE were desired, then the following statements would be sufficient.

```
COMMON I  
DIMENSION I(2,2)
```

(5) Only one COMMON statement may appear per (sub)program.

6.2.1 NUMERICAL COMMON

```
COMMON (n)
```

This statement reserves n storage locations in COMMON; n is any decimal integer constant greater than 0.

Since the amount of COMMON storage is determined by the first COMMON statement in the program, COMMON (n) should be used when any subsequent (sub)programs use more COMMON space than the first encountered (sub) program.

The value of n must be equal to the maximum amount of COMMON space used by the entire program. This value can be computed by allowing three words for each floating point variable and two words for each integer variable.

When COMMON (n) is used, it must be the first statement of the program. A COMMON statement of the type described in section 6.2 can follow COMMON (n). The two types are independent statements.

If COMMON locations are not used and the program does not have an EQUIVALENCE statement, the statement COMMON (0) increases the memory space available to the symbol table generated during program compilation. This statement permits the symbol table to overlay the COMMON and EQUIVALENCE processors.

Example:

```
* MAIN PROGRAM
COMMON (12)
COMMON X, Y, Z
.
.
.
END
SUBROUTINE ONE
COMMON A, B, C, D
.
.
.
END
```

In this example, $n = 12$ because there is a maximum of four variables and each variable occupies three 160-A words. If the COMMON statement in the subroutine had only three variables, a numerical COMMON would be unnecessary.

6.3

EQUIVALENCE

The EQUIVALENCE statement permits variables to share locations in storage. The general format is:

EQUIVALENCE (V_1, V_2), (V_3, V_4), . . .

V_i is a variable or dimensioned variable (integer or floating) written with a single subscript. Elements of a multiply subscripted array must be converted to a singly subscripted variable by the following formulas:

Subscript = $(i) + (j - 1) * I + (k-1) * I + J$ (3 dimensions)

or

Subscript = $(i) + (j - 1) * I$ (2 dimensions)

(See sec, 2.4.1, Storage of Arrays).

A non-subscripted array name is interpreted as the first element of the array.

The EQUIVALENCE statement assigns only the initial locations of the variable pairs. The rest of the array will be stored in consecutive locations.

Example:

```
DIMENSION A(3, 5), M(2, 3, 4)
EQUIVALENCE (A, B), (CC, I (5)), (J(4), M), (A(3), D(2))
```

EQUIVALENCE Rules:

- (1) Within any (sub)programs the EQUIVALENCE statement, as well as the COMMON and DIMENSION statements, are unique to that (sub)program.
- (2) EQUIVALENCE must precede the first executable statement and follow any COMMON or DIMENSION statements.
- (3) Only one EQUIVALENCE statement per (sub)program is allowed.
- (4) Within a (sub)program only the left variable of an EQUIVALENCE pair can occur in a COMMON statement in that (sub)program or previously in the EQUIVALENCE list.
- (5) A formal parameter may not be used in EQUIVALENCE statements.
- (6) (Sub)programs using EQUIVALENCE must precede those not using it since the space for the EQUIVALENCE processor is made available to the compiler as soon as a (sub)program without EQUIVALENCE is encountered; if there are no common locations used, the symbol table will also overlay the common processor.
- (7) The EQUIVALENCE pair may contain any combination of integer floating point variables.

The programmer must remember that an integer takes two 160-A words and a floating point variable requires three 160-A words.

EQUIVALENCE is most commonly used when two or more arrays of different or equal lengths can share the same storage locations.

Example:

```
DIMENSION A(10, 10), I (150)
EQUIVALENCE (A, I)
.
.
.
5 READ 10, A
.
.
.
6 READ 20, I
```

The EQUIVALENCE statement causes arrays A and I to be stored in the same storage locations.

In this example A and I use the same number of 160-A words. Before statement 6 is executed all use of A should be completed. Statement 6 reads the values of I into the storage location previously occupied by A, thus destroying A.

The left variable in an EQUIVALENCE pair determines the starting address to be used by that pair. If the right quantity uses more storage than the left quantity, elements of the right quantity will overlap the locations after the left quantity.

When an EQUIVALENCE statement contains two arrays, the subscript constant of the left quantity must be greater or equal to the subscript constant of the right quantity.

Example:

```
DIMENSION A (10, 10), I (100)
```

```
EQUIVALENCE (A, I (5))
```

This is a programming error that will not give a diagnostic. The first four elements of I (eight 160-A words) are outside of A and overlap some undefined portion of data assignment.

The transfer of data into or out of internal storage is specified by input/output statements. In each statement the programmer specifies, either explicitly or implicitly, the following:

Data List: the data to be moved
Format: the manner in which data will be moved
Input/output process: read, write, punch, print, or type
Input/output device: magnetic tape unit, paper tape unit, card reader, card punch, printer, or typewriter.

The general form of an input/output statement is

NAME n,A,B, . . . ,M

Name specifies the process and input/output device, n references a FORMAT statement which specifies how to move the data, and A,B, . . . ,M are the variables (storage locations) into or out of which the data will be transferred. In binary tape statements no FORMAT statement is necessary.

Example:

READ 22,A,B,I(3,5)

READ	indicates input by cards
22	FORMAT statement number
A,B,I(3,5)	variables (storage locations) into which data from punched cards will be read

WRITE OUTPUT TAPE 3, 10, X, JOE

WRITE OUTPUT TAPE 3	indicates output on magnetic tape 3
10	FORMAT statement number
X,JOE	variables (storage locations) to be written on the tape

7.1

DATA LIST

A data list may contain any number and type of simple or subscripted variables, separated by commas. During an output operation, the contents of storage locations specified by the variables will be transferred to the designated output device. During input, data from an external device will be read into these locations.

Example:

```
PRINT12,A,JOE(1,6),MAX,Y,PUNCH,ZEBRA(12*I-3)
```

Part or all of an array can also be represented as a list item. If an array name appears without a subscript, the whole array is used in the data transfer.

In the example,

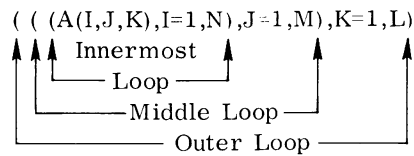
```
.  
. .  
. .  
DIMENSION SAVE (20)  
. .  
. .  
READ 10, SAVE  
. .  
. .  
. .
```

all 20 elements of SAVE will be read into storage locations before the READ statement terminates. All arrays in data lists must be defined in a DIMENSION statement. Data list notation and DO loops provide alternative methods of producing the same results as shown below.

	<u>Data</u>	<u>DO-Implying Form</u>	<u>DO Loop Form</u>
1.	The first N elements of array B	(B(I),I=1,N)	DO 5 I = 1,N LIST (I) = B(I)
2.	All elements of an N row, M column, array B, arranged by rows. (B(1,1),B(1,2) ---B(1,M),B(2,1), B(2,2)--B(N,M))	((B(I,J),J=1,M),I=1,N)	K = 1 DO 5 I = 1,N DO 5 J = 1,M LIST (K) = B(I,J) K = K + 1
3.	All elements of N row, M column, array B, arranged by columns. (B(1,1), B(2,1)--B(M,1),B(2,1), B(2,2)---B(N,M))	((B(I,J),I=1,N),J=1,M) or B (the entire array is transferred)	K = 1 DO 5 J = 1,M DO 5 I = 1,N LIST (K) = B(I,J) K = K + 1
4.	Elements of the second and fifth rows of an N row, M column, array B, arranged by rows.	((B(I,J),J=1,M),I=2,5,3)	K = 1 DO 5 I = 2,5,3 DO 5 J = 1,M LIST (K) = B(I,J) K = K + 1
5.	The first N elements of arrays A and B (A(1),B(1),A(2),B(2)---).	(A(I),B(I),I=1,N)	K = 1 DO 5 I = 1,N LIST = A(I) LIST K = B(I) K = K + 1
6.	All elements of N row, M column, L plane, array B, arranged by planes and columns.	(((B(I,J,K),I=1,N), J=1,M),K=1,L) or B (the entire array is transferred)	KK = 1 DO 5 K = 1,L DO 5 J = 1,M DO 5 I = 1,N LIST (K) = B(I,J,K) KK = KK + 1

The first index variable defined in the list (I in item 6) is stepped first. When it reaches the maximum value it is reset, the next variable to the right is stepped, and the process repeated.

The list forms can be treated like nested DO loops as shown below.



Each loop must be enclosed within parentheses. The right parentheses of each loop except the outer loop must be followed by a comma.

The index variables (N, M, L in the above list) can be either integer variables defined in the program or integer constants. The entire partial array list notation is enclosed in parentheses and separated by commas from the other list variables.

Example:

```
READTAPE 2,BATH,A,( JOE(I,J),J=2,6),I=1,10,2),X(36),(Z(I),I=1,12)
PUNCHFLEX 8,( BTYE(K,J),K=7,M),J=3,12),ORB(6,3),MAX
```

7.2
FORMAT

In all input/output statements except READ TAPE and WRITE TAPE the programmer must specify the type of data (octal integer, decimal integer, floating-point, or alphanumeric) to be stored in each list variable or to be transferred to the output device, and the physical positioning of the data (characters per word or line, spacing between characters, and so forth).

These specifications are written in a FORMAT statement.

```
n FORMAT (s)
```

The statement identifier (section 8.1) referenced by input/output statements is n and (s) is the format specification list. This statement can appear anywhere in the program and can be referenced by any number of input/output statements. A format statement cannot be referenced from outside the subprogram in which it appears. (Variable format control is described in section 7.3.)

Each variable in the data list must have a corresponding specification in the format specification list to indicate the final form of that variable. The mode of the variable must agree with its FORMAT specification. Symbols for spacing and skipping characters, words, or lines are also included in the list.

7.2.1

REPEATING CONVERSION SPECIFICATIONS

The notations nEw.d, nOw, nIw, nFw.d, and nAw, can be used to indicate that the specification is to be repeated n times. A group of specifications can be repeated if they are enclosed in parentheses and preceded by an integer n to indicate the number of repetitions:

```
FORMAT (F6.2, I3, I3, E5.1, F6.2, I3, I3, E5.1)
```

may be written as

```
FORMAT (2(F6.2,2I3, E5.1) ).
```

A repeated parenthetical group may not be contained within another repeated parenthetical group.

It is possible to repeat conversion specifications without using the n factor. The I/O list and format specification need not be the same length. If the original format is exhausted before the end of the input/output list, the remaining data will be converted according to specifications included in the last unquantified parenthetical grouping. Specifications are repeated from the last (left to right) open parenthesis that is not preceded by a repetition factor. The right parenthesis becomes equivalent to a slash.

Example:

```
FORMAT (E12.4, F5.3, (2I3, 3(F6.2) ) )
```

The original format E12.4, F5.3, I3, I3, F6.2, F6.2, is exhausted before the end of the I/O list. Therefore, the remaining data will be converted by I3, I3, F6.2, F6.2, F6.2; (F6.2) was not used for converting remaining data because it is preceded by a repetition factor 3.

7.2.2

FORMAT SPECIFICATIONS

The format specifications in the data list instruct the computer to convert the data from the initial form to the form specified before storing it in the data list variables (input) or transferring it out of internal storage (output). Data conversion is similar for input and output; minor differences will be noted.

Iw Decimal Integer Conversion

I conversion is used to store integer data in integer variables or to output integer data as decimal digits. The field width (number of decimal digits) to be reserved for the item is specified by the unsigned integer constant, w.

On input, a quantity is converted to internal integer representation and is stored right-justified in the specified variable. Blanks and illegal characters are converted to zeros. On output, the contents of the specified variable are converted to BCD decimal digits before they are transferred out of storage. If the field width, *w*, is less than the number of digits in the integer being converted, the integer will be truncated on the right to *w* digits. If the field width, *w*, is greater than the number of significant digits after conversion to BCD, the leading positions are replaced by blanks. Leading zeros are ignored. Since FORTRAN 160-A integers can be no more than 7 digits, I 8 will cover all integer conversions.

<u>Data List</u>	<u>Conversion</u>	<u>Output Data</u>
4650	I4	4650
508763	I5	50876
00308	I5	bb308
76304	I8	bb76304

(b is a blank code)

Example:

```

READ 10, LAMED, MEM, (NUN(I),I=1,8)
10  FORMAT (2I5 , 8I8)

```

The first 5 decimal digits on the card are converted to internal integer format and stored in LAMED.

The second 5 decimal digits on the same card are converted to the internal integer format and stored in MEM.

Each of the next 8 fields of 8 decimal digits is converted to internal integer format and stored in one of the first 8 variables in the array NUN.

CAUTION: Since blanks in a numeric data field are converted to zeros, input specifications must right-justify the input data. For example, if the number 5 is punched in column 3 of a card and the remaining columns are blank, I3 will read this integer as 5, but I4 will read it as 50, I5 as 500, and so on.

Ow Octal Integer Conversion

O conversion is used to store octal quantities or to read out the octal representation of a variable. The field width (number of octal digits) to be reserved for the item is specified by an unsigned integer, *w*.

On input, the octal integer is converted to internal storage format and stored, right-justified in the designated variable. On output, the specified variable is converted to a BCD representation of octal numbers and transferred out of storage. Leading zeros are ignored; and if *w* is less than the number of digits in the quantity being converted, the octal representation is truncated on the right to *w* digits. Blanks and illegal characters are converted to zeros.

<u>Input Data</u>	<u>Conversion</u>	<u>Stored Data</u>
30563	O5	30563
3066bb	O6	306600
4776777	O4	4776
6AQ59	O5	60050

(b = blank)

Example:

```
PUNCH 8, KOPH, MEM, KHET
8  FORMAT (O4,10O6,O5)
```

The variable KOPH is converted to octal digits and punched into the first four columns of a card.

Each of the ten variables in array MEM is converted to octal digits and punched into consecutive six-column fields in the same card. The variable KHET is converted to octal digits and punched into the next five columns in the same card.

The caution given for I conversion applies also to O conversion.

Ew.d Single Precision Floating Point Conversion

E conversion is used for floating point numbers having exponents.

The field width is specified by *w* (digits, plus or minus signs, and E, if present) and *d* specifies the number of digits to the right of the decimal point; *w*, *d*, are unsigned integer constants.

INPUT

E conversion converts the BCD character form of the floating point field to the normalized storage format and stores it in the designated variable. If E conversion is specified, and the input constant contains a decimal point, the decimal point will override the significant digit specification (*d*).

E conversion is used if a +, -, or E is present in the exponent portion of the field. If E conversion is specified and neither +, -, or E is present, the exponent is assumed to be zero and F conversion is used.

If the field width w is larger or smaller (including signs of characteristic and exponent, decimal point, E, and exponent) than the actual length of the input quantities, incorrect values may be read, converted and stored.

Example:

```

Input field      +4.65E+1-2.478E+2
                READ 3, CAT, DOG      |----- 8-----| 9-----|
                3  FORMAT (E8.3, E9.3)  +4.65E+1-2.478E+2
                |----- 6-----| 11-----|
                3  FORMAT (E6.2, E11.4) +4.65E+1-2.478E+2
                |----- 10-----| 7-----|
                3  FORMAT (E10.4, E7.3)  +4.65E+1-2.478E+2
    
```

In memory:

<u>CAT</u>	<u>DOG</u>
.465E02	-.2478E03
.465E01	.0000E38
.465E-11	.478E02

Spaces and illegal characters are converted to zeros. For example, if E7.2 is specified for the constant, 3654E2b (b=blank), then input will be 36.54E20.

OUTPUT

The floating-point number will always be represented by the form

$$\pm.D_1 D_2 \dots D_d E \pm X_1 X_2$$

The field width w must be greater than or equal to the number of significant digits plus six. (The six extra places are for sign of characteristic, decimal point, E, sign of exponent and two digits for the exponent.) The term d cannot be zero.

Floating-point numbers are right-justified in the output field. If the field width w is smaller than the stored number, XXXX. . .X will be output for the entire field. If the field width is larger than necessary, spaces will be provided to the left of the number. Since only 8 significant digits are used in FORTRAN 160-A, the specification E14.8 will cover all floating-point output specifications.

<u>Input Data</u>	<u>Specification</u>	<u>Normalized Input Form</u>
1685062E3	E12.3	.1685062E7
168.5062E3	E12.3	.1685062E6
-39XY2E12	E12.3	-.39002E14
46503E-4	E8.2	.46503E-1
-392077E-3	E10.2	-.392077E1
+ .6217-4	E8.1	.6217E-4
5.3+6	E5.0	.53E7
<u>Stored Value</u>	<u>Specification</u>	<u>Output Form</u>
.650358E6	E12.6	.650358E 06
-.5984E20	E10.4	-.5984E 20

Example:

```

      READ 8, ALEPH,BET
      8  FORMAT (E12.6,E10.4)

```

The floating-point number contained in the first 12 columns of a card will be converted to the internal storage format and stored in the variable ALEPH.

The floating-point number contained in the next 10 columns of the same card will be converted to internal storage format and stored in the variable BET.

Fw.d Floating Point Conversion

The total field width is specified by w; d specifies the number of digits to be retained after the decimal point. F conversion is used for floating-point numbers that do not contain an exponent.

INPUT

The floating-point number is converted to the normalized form of the number and stored in the designated variable. If an E or + or - sign is present after the fractional part of the number, E conversion is used. Spaces and illegal characters are converted to zeros.

The field width w must always be the same as the actual length of the input field containing the input number. If w is too small, only the first w characters will be input. The rest of the number will be converted by the next format specification. If w is larger than the input number, part of the next data field will be included.

Example:

Input field	— 6 — — 7 —		
	-32.54+4.5678		
	READ 3, ALPHA, BETA	<u>ALPHA</u>	<u>BETA</u>
(a)	3 FORMAT (F6.2, F7.2)	-32.54	+4.5678
(b)	3 FORMAT (F4.2, F5.4)	-32.	54+4.
(c)	3 FORMAT (F8.2, F5.3)	-32.54+4	.5678

If the input quantity contains a decimal, d will be ignored.

OUTPUT

The number will be represented by the form

$$(\pm N_1 N_2 \dots N_n . D_1 \dots D_d)$$

unless the number is too large to be expressed by the F format. In such a case E conversion of the form $Ew.(w-6)$ will be used if w is greater than 6. Otherwise XXXX. . .X will be output for the entire field.

<u>Input Form</u>	<u>Specification</u>	<u>Normalized Input Form</u>
-13906	F6.4	-.13906E01
279.370645	F10.4	.27937064E03
5R7E07	F8.2	.507E08
279370645	F8.3	.27937064E06
<u>Stored Value</u>	<u>Specification</u>	<u>Output Form</u>
.1234E4	F7.4	b.1E 04
.1234E4	F6.4	XXXXXX
.1234E4	F10.4	b1234.0000

(b indicates blank)

Example:

```
      READ 8, DALET, HE, (VAV(I),I=1,5)
      8  FORMAT (2F9.3, 5F12.6)
```

The floating numbers in the first two 9-column fields will be converted, retaining 3 digits after the decimal point, and stored in DALET and HE.

The floating point numbers in the next five 12-column fields will be converted to internal storage format and stored in the first five variables in array VAV.

Aw Alphanumeric Conversion

Conversion A stores BCD characters or transfers BCD characters from storage to an output medium. Any legal FORTRAN character will be accepted including blanks.

The number of BCD characters is specified by the unsigned integer constant, w.

The field width w is limited to 6 for floating-point variables and 4 for integer variables. If a floating-point variable is specified, 6 characters will be converted. If a masking operation is specified on a floating point operand, however, only the first 4 characters will be used.

On input, the characters are left-justified and the remaining character positions are filled with the BCD blank character (octal 20). Thus, if A 1 is specified for input to an integer variable location, one character will be left-justified in the variable; the remaining 3 character positions will be filled with the BCD blank character.

<u>Input Data</u>	<u>Specification</u>	<u>Stored Data</u>
RXOP	A3	RXOb
\$) (B	A4	\$) (B
CCCC	A1	Cbbb
1 BA	A3	1bBb

Example:

```
      PUNCH 6, KHIRIK
      6  FORMAT (10A4)
```

Four characters from each of the 10 variables in array KHIRIK will be punched in consecutive 4-column fields in a card.

7.2.3

HEADING AND SPACING SPECIFICATIONS

wHc₁...c_n Heading and Labeling Information

This specification is used to directly output BCD characters included in the format specification list which are generally used for headings and labels. *w* is an unsigned integer specifying the number of BCD characters $C_1 \dots C_n$ in the field.

During output, all characters to the right of *H* are transferred to the specified output device. During input, *n* characters are read into the *H* specification. These may be any legal FORTRAN character including blank.

Example:

```
PUNCH 5, LAMED
5  FORMAT (35H TWAS BRILLIG AND THE SLITHY TOVES, I8)
```

The first 35 columns of a card will be punched with the characters TWAS BRILLIG AND THE SLITHY TOVES

Columns 36-43 will be punched with the contents of variable LAMED (in decimal digits).

wX Intra-line Spacing

With this specification characters may be skipped during input or blanks inserted between characters during output; *w* is an unsigned integer.

During input, *wX* specifies that the next *w* characters are to be ignored. During output, *wX* specifies that *w* BCD blanks are to be inserted on the output record before the next variable is transferred from storage.

Example:

```
READ 5, IC, MIN, ME
5  FORMAT (A4,10X,I8,4X,A4)
```

The first 4 BCD characters on the card are read into variable IC. The next 10 columns on the card are skipped and the digits in columns 15-22 are stored in MIN. The next 4 columns are skipped and the BCD characters in columns 27-30 are stored in variable ME.

/ Inter-spacing of Records

The symbol / signals the end of a BCD record; it can be used to skip lines, cards, or magnetic tape records. During input, / specifies that control passes to the next record or card.

Example:

```
READ 6, (HE(J), 40), BET
6  FORMAT (40A1/F12.6)
```

The first 40 BCD characters in the first card are read into the first 40 variables in array HE, one character left-justified in each variable. The remainder of the card is ignored and the floating point number in the first 12 columns of the next card is converted to internal storage form and stored in BET.

During output, / signals the end of one record and the beginning of a new line, record, or card.

Example:

```
A = 25.3          PRINT 11, A, B
B = 25.3          11  FORMAT (6HSUBTOT, 6X, F4.2//6HTOTAL,
                          6X, F4.2)
```

// will cause the printer to double space before printing a new record.

Print-Out	SUBTOT	25.3	line 1
			line 2
	TOTAL	25.3	line 3

Each line corresponds to a BCD record; line 2 is a null record.

7.24

COMPLETE FORMAT

SPECIFICATIONS

All individual conversion specifications and the entire specification list is included in parentheses in the FORMAT statement. A comma after the H specification will be ignored by the compiler and can be omitted. A / replaces a comma and need not be separated by commas from other specifications. A FORMAT statement may contain only Hollerith information or spacing specifications. Any FORMAT statement can be referenced by more than one input/output statement. FORMAT statements are local to the subroutines in which they appear.

Source Program Code to Read and Store Format

```
DIMENSION K(100)
3  FORMAT (4A4)
   READ 3, (K(I), I = 1,4)
```

Input Data

(E10.3,F4.2,2I4)

K must have enough space reserved by the DIMENSION statement to provide for any FORMAT specification list anticipated. Similarly, the A4 specification in the FORMAT statement must be adequate to store the specification list as it is read in. (Blanks are ignored except in H specifications.) The READ statement specifies that data will be stored two characters per 160-A word in BCD code. After execution, array K will contain the following:

	<u>Word</u>	<u>Characters</u>	<u>BCD Code</u>	
K (1)	1	(E	3465	Each integer K (I) contains two 160-A words.
		10	0112	
	.	.	.	
K (4)	7	2I	0271	
		4)	0474	
	.	.	.	

Later in the source program, an input or output statement can refer to K as its FORMAT statement designator. Conversion of the input/output list will proceed in accordance with the specification list previously read into K; for instance:

```
   READ K, BAILLY, YGUEM, LATOUR, MOUTON
```

Data will be read into BAILLY under the E10.3 conversion, into YGUEM under the F4.2 conversion, and into LATOUR and MOUTON under the I4 conversion. K cannot be subscripted.

7.4

MAGNETIC TAPE STATEMENTS

In 160-A installations, magnetic tape units are integral logical tape unit numbers, 1,2, . . . n, where n is the number of available units. To allow the programmer and the operator latitude in the selection of tape units for a particular program, digits are not assigned on an absolute basis, but rather, on a logical basis. Thus, the programmer selects any of the tape unit numbers used in input/output statements; and at execution time, the operator is told what numbers to assign to the different units. For example, if the program uses two data tapes, A and B, the programmer can assign A to tape unit 1 and B to tape unit 2. The operator will be told that the unit on which he loads data tape A will be designated as unit 1, and the unit on which he loads data tape B will be unit 2.

In the following statements, *i* = logical tape unit number, *n* = FORMAT statement number, *A* = data list.

READ INPUT TAPE *i*, *n*, *A*

Data is read from tape *i*, converted according to FORMAT statement *n*, and stored in *A*.

WRITE OUTPUT TAPE *i*, *n*, *A*

Data from *A* is converted according to FORMAT statement *n*, and written on tape unit *i*.

READ TAPE *i*, *A*

Data from tape unit *i* is read in binary format into *A*. This statement is used to read data written by the WRITE TAPE statement, and has no format statement designator because it transfers data in binary format only.

WRITE TAPE *i*, *A*

Data from array *A* is written on tape unit *i* in binary format. Since only binary format is used with this statement, no format statement is referenced.

BACKSPACE *i*

This statement backs up BCD tape *i* one physical record, or binary tape *i* one logical record. (Appendix E)

REWIND *i*

This rewinds tape *i* to load point, but does not disconnect the tape from the system. The tape is available for further use.

ENDFILE *i*

This statement writes an end of file mark on tape *i*.

7.5 PUNCHED CARD STATEMENTS

In the following statements, *n* = FORMAT statement number, *A* = data list.

READ *n*, *A*

Data is read from punched cards, converted according to FORMAT statement *n*, and stored in *A*. Illegal BCD characters are converted to blanks during processing and further converted to zeros if they occur in a numeric data field. The maximum record length is 80 card columns.

PUNCH n,A

Data stored in A is converted according to FORMAT statement n, and punched into cards. All 80 card columns can be used.

7.6

FLEXOWRITER STATEMENTS

Flexowriter characteristics are discussed in Appendix F. In the following statements, n = FORMAT statement number, A = data list.

READ FLEX n, A

Data from paper tape prepared on a Flexowriter is converted according to FORMAT statement n and stored in A. Conversion begins with the first character on the tape.

PUNCH FLEX n,A

Data from A is converted according to FORMAT statement n and punched into paper tapes for subsequent listing on the Flexowriter.

7.7

TYPEWRITER STATEMENTS

In the following statements, n = FORMAT statement number, A = data list.

READ TYPE n,A

Information from the typewriter is converted according to FORMAT statement n and stored in A. Before each record is read this statement returns the carriage, types a question mark, ?, and sets the typewriter to lower case. After the dash is typed, the operator types the data, one record at a time. Each typed line is recognized as a record and can have a maximum of 120 characters. A record ends when 120 characters are typed or when a carriage return is struck. Since there are less than 120 character positions on a typewriter line, the typewriter carriage must be returned manually to obtain 120 characters on a single record.

WRITE TYPE n,A

Information from A is converted according to FORMAT statement n and is typed out on the typewriter. When the typewriter is selected for output, the system automatically performs a typewriter carriage return before each data record. Illegal BCD characters are typed as blanks except in numeric data fields where they are typed as zeros. Further information on the typewriter appears in Appendix F.

7.8

PRINTER STATEMENTS

PRINT n,A

Data in A is converted according to FORMAT statement n and printed. The maximum record length is 121 characters, but the first character of every record is used for carriage control on the standard CONTROL DATA® printer and is not printed. Carriage control symbols are given below.

BCD 1	PAGE EJECT	A record which has a BCD 1 as the first character will appear at the top of a new page.
BCD 0	DOUBLE SPACE	This code will cause a double space before printing.
Any other BCD Characters (normally blank)	SINGLE SPACE	This code will cause a single space after printing a line.

There is no automatic page eject on program controlled output. A BCD 1 must appear as the first character of any line which begins at the top of a new page.

7.9

SPECIFICATIONS FOR NON-STANDARD EQUIPMENT


Each 160-A installation has a systems tape which describes the standard input/output equipment used by the system. If, however, a program uses a non-standard input/output device, standard equipment assignments must be changed on the systems tape and each input/output statement must reference a subroutine on the systems tape which uses non-standard equipment. This subroutine is referenced by including the subroutine name in parentheses after the input/output statement name.

Example:

```
READ (HOOKUP) 6,A,B,C,JOE  
WRITE OUTPUT TAPE (HOOKUP) 3,5,MAX,ETC
```

(HOOKUP is a program for handling non-standard equipment.)

FORTRAN 160-A coding forms contain 80 columns in which the FORTRAN 160-A characters are written, one character per column.

1604 FORTRAN CODING FORM			NAME
PROGRAM	PAGE		
ROUTINE	DATE		
		FORTRAN STATEMENT	
STATEMENT NO.	COLUMN	0 - ZERO # - ALPHA 0	1 - ONE I - ALPHA I
		2 - TWO # - ALPHA Z	SERIAL NUMBER
1	1	PROGRAM, COMPARE,	
C	1	COMPARE, EFFECTIVE, SPEED, OF, 3600, TO, 7094, I, I,	
	1	DIMENSION, LOMND(40), PRCNT(20), ACOM(20), BCOM(20),	
	1	AANS(20), BANS(20),	
	1	READ, 15, (LOMND(2*I-1), LOMND(2*I), PRCNT(I), ACOM(I), BCOM(I), I=1, 20),	
	1	FORMAT(2A4, 3F6.2),	
	1	READ, 25, ACYC, BCYC,	
	1	FORMAT(2F10.2),	
	1	A TOT = 0.	
	1	B TOT = 0.	
	1	DO, 30, I = 1, 20,	
	1	AANS(I) = ACOM(I) * ACYC * PRCNT(I),	
	1	BANS(I) = BCOM(I) * BCYC * PRCNT(I),	
	1	A TOT = A TOT + AANS(I),	
	1	B TOT = B TOT + BANS(I),	
	1	CONTINUE	
	1	PRINT, 40, (LOMND(2*I-1), LOMND(2*I), AANS(I), BANS(I), I=1, 20),	
	1	FORMAT(20X, 2A4, 20X, F10.2, 20X, F10.2),	
	1	PRINT, 50, A TOT, B TOT,	
	1	FORMAT(/ / 42X, 6HTOTAL = F10.2, / 42X, 6HTOTAL = F10.2),	
	1	END.	

8.1 STATEMENTS

The statements, instructions and descriptions in the FORTRAN 160-A language are written in columns 7 through 72. Each statement must begin on a new line and, although there can be no more than one statement to a line, any statement may extend over additional lines. Because blanks are ignored, they may be used freely in any FORTRAN statement. Blanks are significant only in an H field of a FORMAT statement specification list.

If the type designator (B or C) is required, it is written in column 1 of the first line of the statement. B indicates a masking expression and C identifies a comment.

Any statement may have an identifier, but only those statements referred to elsewhere in the program require identifiers. An identifier is a string of from 1 to 5 digits. If there is a type designator (B or C) in column 1, the

identifier is limited to columns 2 through 5, otherwise it can occupy columns 1 through 5. Statement numbers need not be in any sequence, but within the main program or within any subroutine, no two statements may have the same number.

8.2

CONTINUATION

The first line of every statement must have a blank or zero in column 6. If the statement occupies more than one line, all subsequent lines of a statement must have a FORTRAN character other than blank or zero in column 6. The end of a card does not act as a blank. Therefore, if a statement is continued beyond a line and a blank is to appear after column 72, this blank must appear in column 7 of the next line.

8.3

COMMENTS

Comments can be included in the program by placing the type designator C in column one on the coding sheets. Comments will be ignored by the compiler. They can extend from columns 2 to 72; any keypunch character may be used in the comments field.

8.4

IDENTIFICATION FIELD

Columns 73 through 80 are ignored in the translation process. They may be used for identification when the program is put on punched cards.

8.5

PUNCHED CARDS

Each line of the coding form corresponds to one 80-column card, and the terms line and card are often used interchangeably. Source programs and data can be read into the computer from cards; an object program memory map, program diagnostics, and data can be written directly on cards. The object program can be read out on magnetic tape but not punched directly on cards.

A blank card included in the source program will be read, and it will cause compilation errors. However, one blank card must follow the source program END card to signal an end to compilation.

If punched cards are used for data input, the data cards follow the blank card at the end of the source deck. All 80 columns can be used for data input.

8.6

PAPER TAPE

Punched paper tape prepared on a Flexowriter can be used for source program and data input.

If the 160-A FORTRAN source program is prepared on the Flexowriter, a 72-character line of input constitutes a maximum length record. A carriage return (CR) on the Flexowriter terminates a record. Fields within a record are fixed exactly as on punched cards. On source code input, the statement field can be terminated by a carriage return at whatever point the source code terminates within the field. When a tab is encountered, it is interpreted as an information error; control is returned to column seven and the erroneous information is destroyed. Delete characters, carriage returns, and case codes do not count as spaces in a field.

On data input, an 80-character line can be used. A tab is an illegal character and is replaced with a blank.

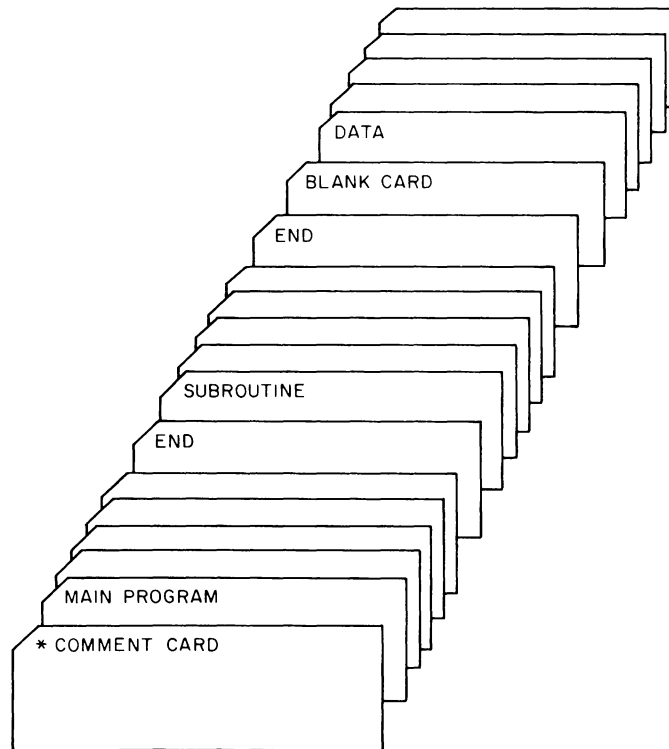
A blank record must follow the final source program END statement. This punch pattern can be created on the paper tape by two successive carriage returns. This record pattern signals an end to the compile process; it should appear only after the terminal END statement. Appendix F includes additional information on paper tape.

8.7

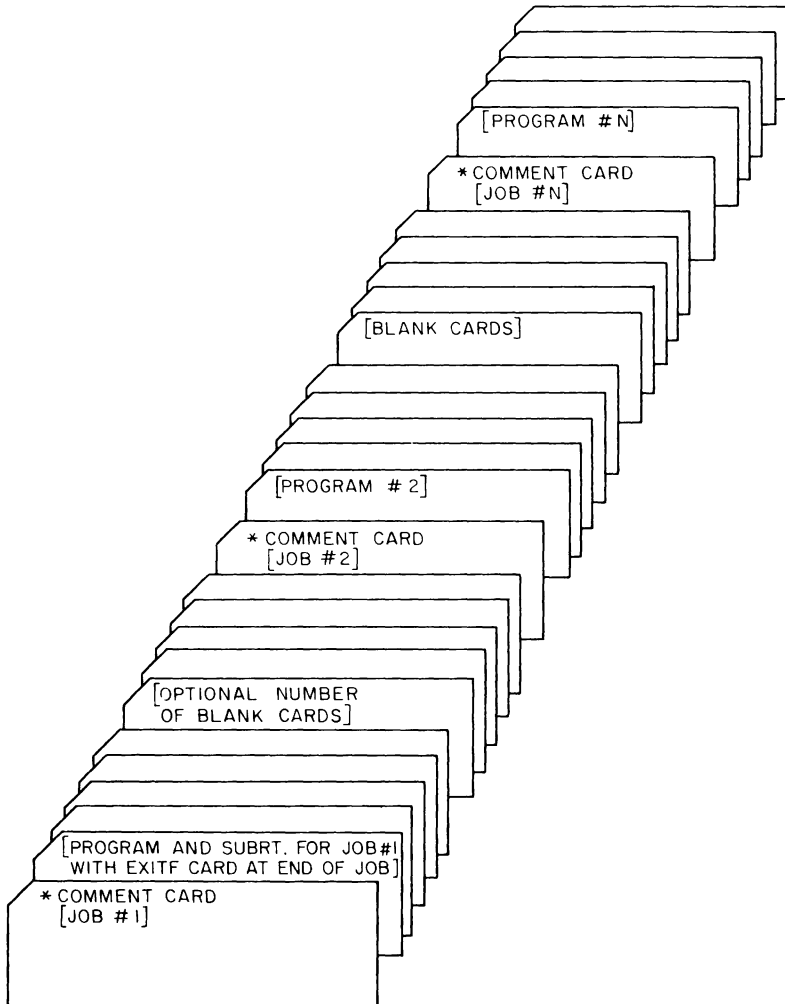
MAGNETIC TAPE

Two tapes are necessary for the 160-A FORTRAN system. One tape holds the system library tape; the second tape holds the compiled program. If the source program is loaded from tape, a third tape is required. Diagnostics and an object program memory can be output on magnetic tape.

The bootstrap routine requires that each job begins with a card which has an asterisk in column 1. The remaining columns of this card may contain comments. Each asterisk control card is output to the standard output device.



Single Job



Stacked Jobs

Example One:

The "least squares" technique applies to the line, $Y_i = ax_i + b$, and involves solving normal equations:

$$Na + b \sum X_i - \sum Y_i = 0$$

$$a \sum X_i^2 + b \sum X_i - \sum X_i Y_i = 0$$

N = number of paired points, X_i and Y_i
 Summations (\sum) shown are from

$$i = 1 \text{ to } i = N$$

Therefore, if given paired points:

$$X_1 = 0.50, Y_1 = 0.38$$

$$X_2 = 1.00, Y_2 = 0.82$$

$$X_3 = 2.50, Y_3 = 2.00$$

The results are

$$N = 3, \sum X_i = 4, \sum Y_i = 3.20$$

$$\sum X_i^2 = 7.50, \sum X_i Y_i = 6.01$$

and the normal equations are

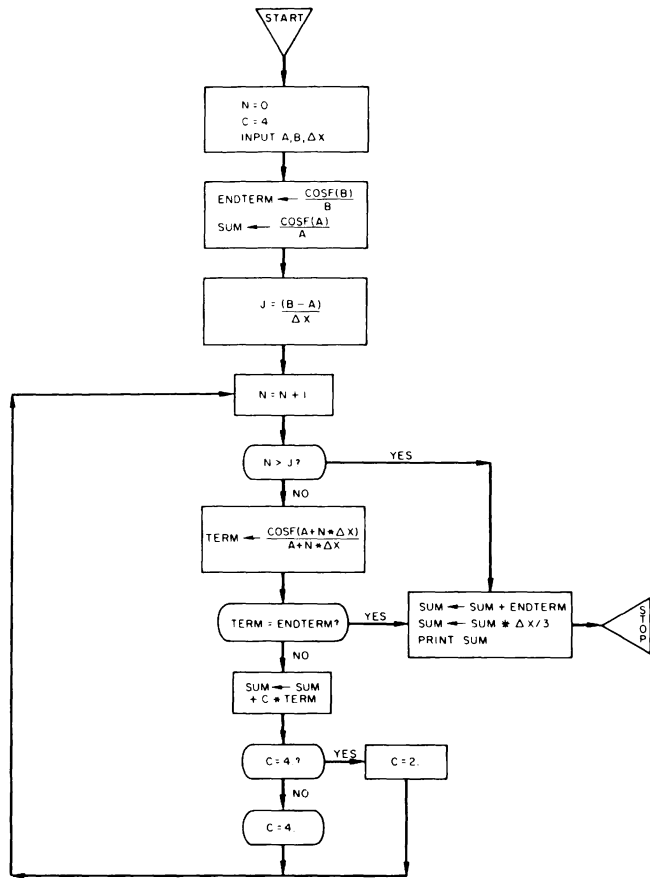
$$\begin{cases} 3a + 4b - 3.2 = 0 \\ 4a + 7.5b - 6.01 = 0 \end{cases}$$

Normal equations can be solved for x and y by using determinants.

$$\begin{cases} ax + by - c = 0 & \text{is solved} & x = \frac{-ce + bf}{ae - bd} \\ dx + ey - f = 0 & \text{by} & y = \frac{af + cd}{ae - bd} \end{cases}$$

X and Y values are read in by groups. Each group is preceded by a control card indicating the number, N , of paired points in the group. A control card, $N = 0$, indicates the end of the data. N can vary from 2 to 50; X and Y values contain three significant digits. Write a program which will print out a and b in a form similar to

$$A = \quad \wedge \wedge \wedge \wedge \wedge \quad B =$$



```

* EXAMPLE PROGRAM ONE
  DIMENSION X(50),Y(50)
10 READ 1, N
  1 FORMAT(12)
    IF(N) 2,6,2
  2 READ 3,(X(I),Y(I),I=1,N)
  3 FORMAT(2E9.3)
    SUMX=0.
    SUMY=0.
    SUMXY=0.
    SUMX2=0.
    DO 4 I=1,N
      SUMX=SUMX+X(I)
      SUMY=SUMY+Y(I)
      SUMXY=SUMXY+X(I)*Y(I)
  4 SUMX2=SUMX2+X(I)**2
    FN=N
    A=(SUMY*SUMX2-SUMXY*SUMX)/(FN*SUMX2-SUMX**2)
    B=(FN*SUMXY-SUMX*SUMY)/(FN*SUMX2-SUMX**2)
    PRINT 5, A,B
  5 FORMAT(3H A=, E9.3,5X,2HB=, E9.3)
    GO TO 10
  6 CONTINUE
  END
  
```

MEMORY MAP

INTEGER VARIABLES

IDENT	LEVEL	OBJECT CODE	LOCATION	UP	SUBROUTINE
I	1	1	7307	1	1057
N	1	1	7313		NOT USED
IO	0	1	7775	1	1065

FLOATING POINT VARIABLES

IDENT	LEVEL	OBJECT CODE	LOCATION
B	1	1	7251
A	1	1	7254
FN	1	1	7257
SUMX2	1	1	7264
SUMXY	1	1	7267
SUMY	1	1	7272
SUMX	1	1	7300

FLOATING POINT ARRAYS

IDENT	LEVEL	OBJECT CODE	LOCATION	DIMENSION	DIM1	DIM2
Y	1	1	7315	1		
X	1	1	7543	1		

CONSTANTS

VALUE	LEVEL	OBJECT CODE	LOCATION
0.0	2	1	7262
		1	7275
	50	1	7771
	1	1	7773

STATEMENT NUMBERS

IDENT	LEVEL	OBJECT CODE	LOCATION
4	1	1	0720
6	1	1	1041
2	1	1	0566
10	1	1	0547
0	1	1	0547

FORMAT STATEMENTS

IDENT	LEVEL	OBJECT CODE	LOCATION
5	1	1	7233
3	1	1	7303
1	1	1	7311

LIBRARY FUNCTIONS

IDENT	LEVEL	OBJECT CODE	LOCATION
(P1660		0	7130
(R1671		0	7240
INPUT		1	0001
A**I		1	0457

ERASABLE STORAGE 1 1075 TO 1 7233
A=-.615E-02 B= .804E 00

Example Two:

Simpson's rule for approximating a definite integral is:

$$\int_a^b f(x)dx = \frac{\Delta x}{3} (f(a) + 4f(a+\Delta x) + 2f(a+2\Delta x) + 4f(a+3\Delta x) + \dots + f(b))$$

For example, using $a = 1$, $b = 2$, $\Delta x = 0.25$, and the integral $\int_1^2 \sqrt{1+4x} dx$ gives:

$$\begin{aligned} f(a) &= f(1) = \sqrt{1+4(1)} = \sqrt{5} \\ 4f(a+\Delta x) &= 4f(1+.25) = 4f(1.25) = 4\sqrt{1+4(1.25)} = 4\sqrt{6} \\ 2f(a+2\Delta x) &= 2f(1+2(.25)) = 2f(1.5) = 2\sqrt{1+4(1.5)} = 2\sqrt{7} \\ 4f(a+3\Delta x) &= 4f(1+3(.25)) = 4f(1.75) = 4\sqrt{1+4(1.75)} = 4\sqrt{8} \end{aligned}$$

Since $(a + 4\Delta x) = 1+4(.25) = 2 = b$; the last term is $f(2) = \sqrt{1+4(2)} = \sqrt{9}$.

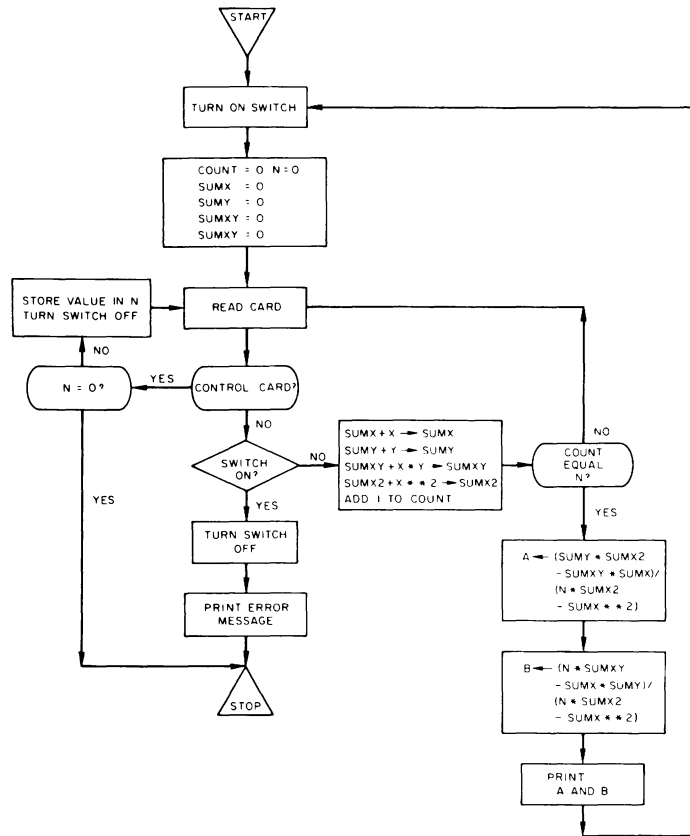
The last term is reached when $(a+n\Delta x) = b$, and no number (2 or 4) appears in front of the first or last terms.

The sum of the equation is:

$$\int_1^2 \sqrt{1+4x} dx = \frac{.25}{3} (\sqrt{5} + 4\sqrt{6} + 2\sqrt{7} + 4\sqrt{8} + \sqrt{9}) = 2.636 \text{ approx.}$$

Card 1 contains the value Δx , ranging from $-.25$ to $+.25$. The first two fields of Card 2 contain a and b , each of which range from -9.99 to $+9.99$. Write a program which will use Simpson's rule to approximate and print the results for the integral,

$$\int_a^b \frac{\cos x}{x} dx$$



```

* EXAMPLE PROGRAM TWO
  READ 10, DELTAX, A, B
10  FORMAT(E8.2/2E9.3)
  ENTERM=COSF(B)/B
  SUM=COSF(A)/A
  J=(B-A)/DELTAX
  C=4.
  I-J+1
  DO 14 N=1, I
  FN=N
  IF(FN-J)9,9,17
9  TERM=COSF(A+FN*DELTAX)/(A+FN*DELTAX)
  IF (TERM-ENTERM)16,17,16
16  SUM=SUM+C*TERM
  IF(C-4.) 13,12,13
12  C=2.
  GO TO 14
13  C=4.
14  CONTINUE
17  SUM=SUM+ENTERM
15  SUM=(SUM*DELTAX)/3.
  PRINT 20, SUM
20  FORMAT(5H SUM=, E9.3)
  END
  
```

MEMORY MAP

INTEGER VARIABLES

IDENT	LEVEL	OBJECT CODE	LOCATION	P	SUBROUTINE
N	1	1	7732		NOT USED
I	1	1	7734		NOT USED
J	1	1	7744		NOT USED
IO	0	1	7775	0	20

FLOATING POINT VARIABLES

IDENT	LEVEL	OBJECT CODE	LOCATION
TERM	1	1	7724
FN	1	1	7727
C	1	1	7741
SUM	1	1	7746
ENTERM	1	1	7751
B	1	1	7762
A	1	1	7765
DELTA	1	1	7770

CONSTANTS

VALUE	OBJECT CODE	LOCATION
0.30000000 E 01	1	7716
0.20000000 E 01	1	7721
0.40000000 E 01	1	7736
1	1	7773

STATEMENT NUMBERS

IDENT	LEVEL	OBJECT CODE	LOCATION
15	1	1	0370
12	1	1	0340
13	1	1	0346
16	1	1	0317
17	1	1	0362
9	1	1	0265
14	1	1	0352
0	1	1	0161

FORMAT STATEMENTS

IDENT	LEVEL	OBJECT CODE	LOCATION
20	1	1	7706
10	1	1	7754

LIBRARY FUNCTIONS

IDENT	LEVEL	OBJECT CODE	LOCATION
(P1660		0	6712
(R1671		0	7020
INPUT		0	7136
SINF		1	0001
COSF		1	0151
ERASABLE STORAGE		1 0432 TO 1	7706
SUM=	.165E 02		

APPENDIX SECTION

UTILITY FUNCTIONS

Utility functions have been added to the 160-A FORTRAN source language to perform the following operations:

1. Check for an end of file on a read operation. (XEOF)
2. Return control from a running program to the compiler. (XEXITF)
3. Dump memory for later execution. (XPDMPF, X163F, XCDMPF, X1607F)
4. Plot lines on the 165 Plotter. (PLOTf)

All except the XEOF and PLOTf functions are written in the form:

$$I = \text{function } (P)$$

I is any integer variable and P is the parameter list. All functions except the PLOTf function use a single parameter. The PLOTf function has the form:

$$X = \text{function } (P)$$

X is a floating point variable and P is the parameter list. The variable I or X is a dummy variable required by the form of the statement; the contents of I or X will always be altered when the function is executed.

XEOF FUNCTION The XEOF function tests whether or not an end of file was detected during the last executed READ TAPE or READ INPUT TAPE statement. This function is used as an operand in an IF statement.

$$\text{IF } (\text{XEOF}(V)) n_1, n_2, n_1$$

V is any fixed or floating point variable and n_1 and n_2 are statement numbers. Control will transfer to n_1 if an end of file mark was read during the last tape-read operation. Otherwise, control transfers to n_2 . The contents of V are not changed when the statement is executed.

If an end of file is detected during a read operation, zeros are stored in all variables contained in the input list, regardless of whether or not an XEOF test is made by the programmer.

Example:

```
      .  
      .  
      .  
      READ INPUT TAPE 2, 10, A, B, C, N  
      IF (XEOF (DUM) ), 7, 8, 7  
7     PRINT 3  
3     FORMAT (17H END OF FILE ON 2)  
      STOP 77  
8     DO 11 I = 1, N  
      .  
      .  
      .
```

If an end of file is detected during a read operation, the program prints END OF FILE ON 2 and stops. Otherwise, processing continues with statement 8.

XEXITF FUNCTION This function transfers control from the execution of one program back to an initialization routine within the system for compilation of the next program.

I = XEXITF (N)

I is any integer variable and N specifies the magnetic tape unit as follows:

<u>N</u>	<u>magnetic tape unit</u>
0	163 or 162
1	1607

The XEXITF function statement should replace all STOP statements in a program.

**XPDMPF,X163F,
XCDMPF,X1607F
FUNCTIONS**

These functions are used to obtain reloadable core dumps. XPDMPF outputs to paper tape, X163F outputs to 163 or 162 magnetic tape unit 2, XCDMPF outputs to cards, using the 523 card punch and X1607F outputs to 1607 magnetic tape unit 2.

I = XCDMPF (N)
I = XPDMPF (N)
I = X163F (N)
I = X1607F (N)

I is any integer variable and N is the number of the last bank to be dumped. N can be an integer variable or constant; I is a dummy variable whose value will be changed during execution. A halt occurs after the dump is completed. Running from this halt continues execution at the statement following the dump function.

Separate loaders are available for each of the dumps. Operating instructions for these loaders are contained in the 160-A FORTRAN OPERATIONS MANUAL. When the dump is reloaded, execution begins at the statement following the dump function.

```

*   THIS IS A PROGRAM TO DEMONSTRATE THE X163F FUNCTION
    DIMENSION SIN(360), COS(360), X(360)
    PI = 3.14159
    DO 10 I = 1,360
    X(I) = I*PI/180.
    SIN(I) = SIN(X(I))
10   COS(I) = COS(X(I))
    R=XPDPF(1)
    PRINT 100
100  FORMAT (52H DEGREE X(I) COS(I) SIN(I),/)
    DO 20 I = 1,360
    PRINT 200, I,X(I),COS(I),SIN(I)
200  FORMAT (110,5X,F10.8,4X,F10.8,4X,F10.8)
    20   CONTINUE
    X=EXITF(0)
    END

```

PLOTF FUNCTION The PLOTF function provides output on the 165 plotter.

$$X = \text{PLOTF}(A,B,J)$$

X is a floating point variable and A and B are floating point variables or constants, interpreted according to the value of J. J may be an integer constant or variable.

Upon return from the function, X contains the value of the 12 switches on the 165 plotter. These can be tested using a masking statement.

Example:

```

.
.
.
X = PLOTF (1,1,1)
B IF (X*4) 20,10,20
.
.
.

```

If the third switch from the right is set, control transfers to statement 20.

If J = 1, A and B define the coordinate scale factors in units per inch for subsequent PLOTf statements. If no scales are given, the scale factors are initially set to 1.

If J = 2, A and B define the coordinates of the present position of the pen in the units defined by a J = 1 PLOTf statement. A and B are initially set to 0; all following coordinates of PLOTf functions which control pen motion will be interpreted relative to these initial coordinates. If a specific initial position on the plotter is desired, the operator must manually place the pen at that point. If an initial pen position is not defined before the first pen motion is requested, the relative coordinates are 0, 0.

If J = 3, the pen is raised from the paper, moved to position (A,B) and lowered on the paper. The direction and length of pen movement is determined as shown below. A_1 and B_1 are the present coordinates of the pen (defined either by a PLOTf ($A_1, B_1, 2$) function or by a previous pen motion, and A_2, B_2 are given in the PLOTf ($A_2, B_2, 3$) function: AS and BS are the scale factors.

Pen motion in the A direction: *✓* *AS & BS = 10*

$$\Delta A = \frac{100}{AS} (A_2 - A_1) \text{ in plotter units (1 plotter unit} = \frac{1}{100} \text{ inch)}$$

no hundred if inches

Pen motion in the B direction: *✓*

$$\Delta B = \frac{100}{BS} (B_2 - B_1) \text{ in plotter units}$$

no hundred if inches

•• Direction = $\tan^{-1} \Delta B / \Delta A$ degrees

$$\text{Length} = ((\Delta B)^2 + (\Delta A)^2)^{1/2} \text{ plotter units}$$

All lengths are rounded to the nearest $\frac{1}{100}$ inch.

If J = 4, motion is the same as J = 3, except that the pen remains in contact with the paper at all times. The pen is assumed to be down.

If J is any integer other than 1, 2, 3, or 4, it will be treated as though J = 4. A single PLOTf function may not request pen motion in excess of 20 inches in either A or B direction.

Programming

A plot subprogram should begin with statements (A, B, 1) and (A, B, 2) to define the scale factors and initial position of the pen.

Example:

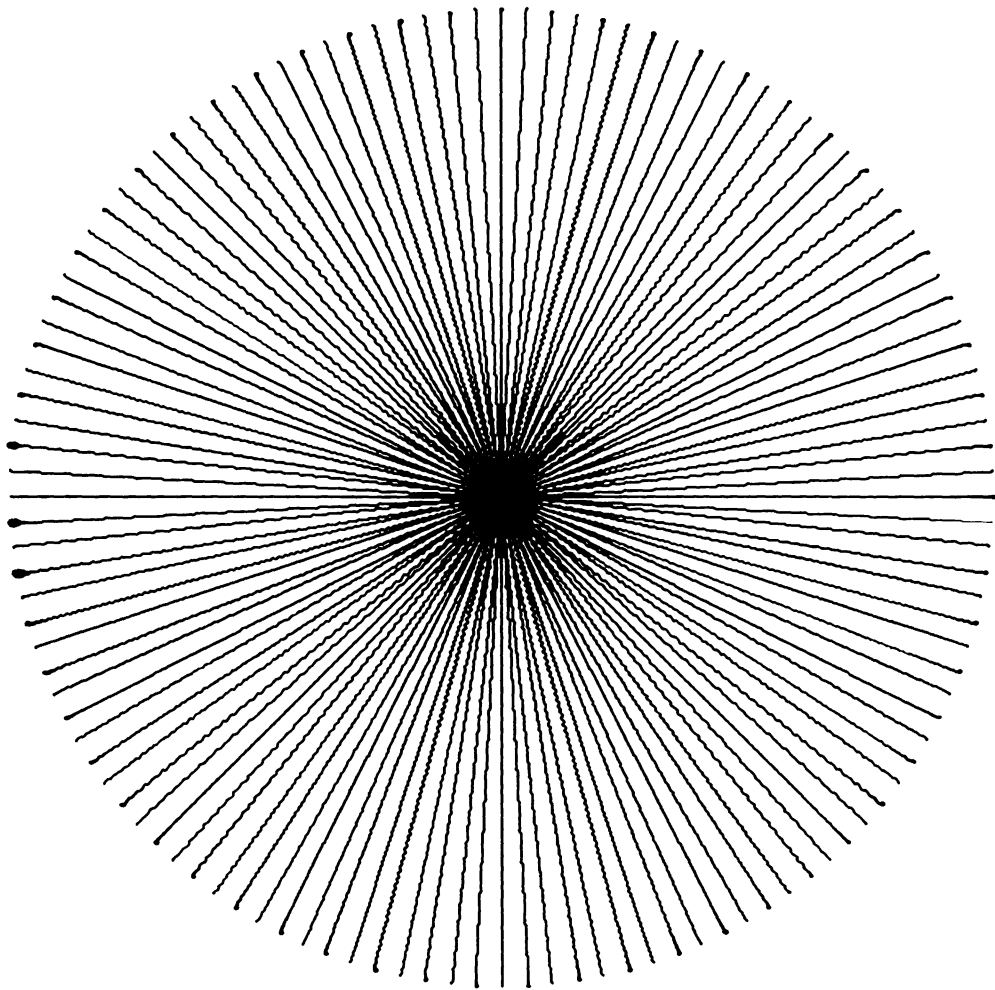
The following program produces the patterns shown on pages 85-87.
The variable R is the radius and may be changed to give a larger or smaller plot.

```
C      160-A PLOT DEMONSTRATION-PRODUCES THREE PLOTS EACH 10
      INCHES WIDE
C      POSITION PLOTTER PEN ON LEFT HAND SIDE OF 165 PLOTTER
      CALL RAY
      CALL KRIECH
      CALL LAMP
      END
      SUBROUTINE RAY
C      PRODUCES DIAMETERS OF A CIRCLE, EACH SPACED 3 DEGREES
      APART
      R=5.0
      4  FORMAT (2(4X,E14.8),4X,12,2(4X,E14.8))
      Q=PLOT(1.0,1.0,1)
      Q=PLOT(-R,0.0,2)
      N=1
      T=3.1415926/2
      Z=3.1415926/60
      DO 10 I=1,60
      T=T-Z
      XR=R*COS(T)
      YR=R*SIN(T)
      XL=-XR
      YL=-YR
      GO TO (20,30) N
20  N=2
      PRINT 4,XR,YR,N,XL,YL
      Q=PLOT(XR,YR,3)
      Q=PLOT(XL,YL,4)
      GO TO 10
30  N=1
      PRINT 4,XR,YR,N,XL,YL
      Q=PLOT(XL,YL,3)
      Q=PLOT(XR,YR,4)
10  CONTINUE
      Q=PLOT(-R,15.0,3)
      END
      SUBROUTINE KRIECH
C      CONCENTRIC CIRCULAR ENVELOPES
      DIMENSION X(20),Y(20)
      R=5.0
      NPOINT=19
      PI=3.1415926
      DELTA=2*PI/NPOINT
```

```

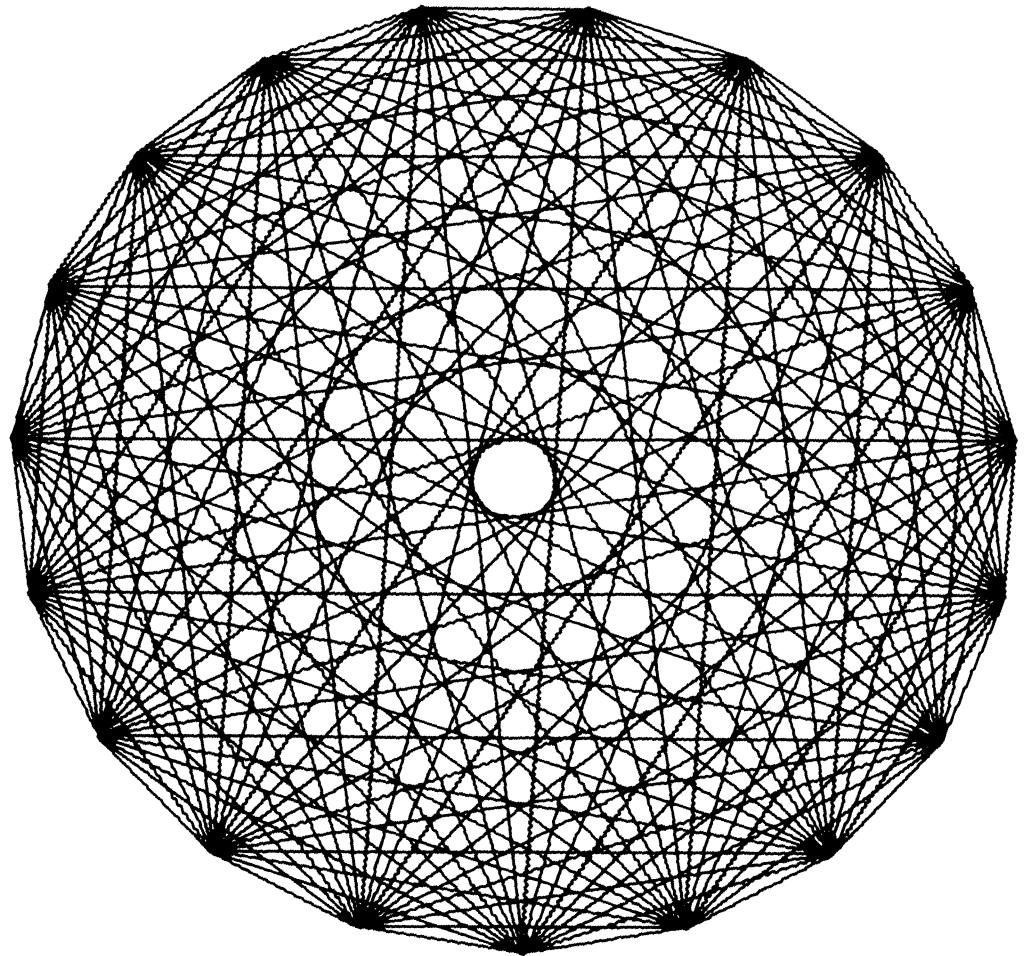
DO 101 I=1,NPOINT
PI=PI-DELTA
X(I)=R*COSF(PI)
Y(I)=R*SINF(PI)
101 CONTINUE
Q=PLOT(1.0,1.0,1)
Q=PLOT(-R,0.0,2)
Q=PLOT(X(I),Y(I),3)
LIMIT=NPOINT/2
L=1
DO 10 J=1,LIMIT
DO 10 K=1,NPOINT
L=L+J
IF(L-NPOINT)10,10,3
3 L=L-NPOINT
10 Q=PLOT(X(L),Y(L),4)
Q=PLOT(-R,15.0,3)
END
SUBROUTINE LAMP
R=5.0
Q=PLOT(1.0,1.0,1)
Q=PLOT(-R,0.0,2)
N=1
T=3.1415926/2
Z=3.1415926/60
DO 10 I=1,120
T=T-Z
XR=R*COSF(T)
YR=R*SINF(T)
XL=-XR
YL=0.0
GO TO (20,30) N
20 N=2
Q=PLOT(XR,YR,3)
Q=PLOT(XL,YL,4)
GO TO 10
30 N=1
Q=PLOT(XL,YL,3)
Q=PLOT(XR,YR,4)
10 CONTINUE
Q=PLOT(-R,15.0,3)
END

```

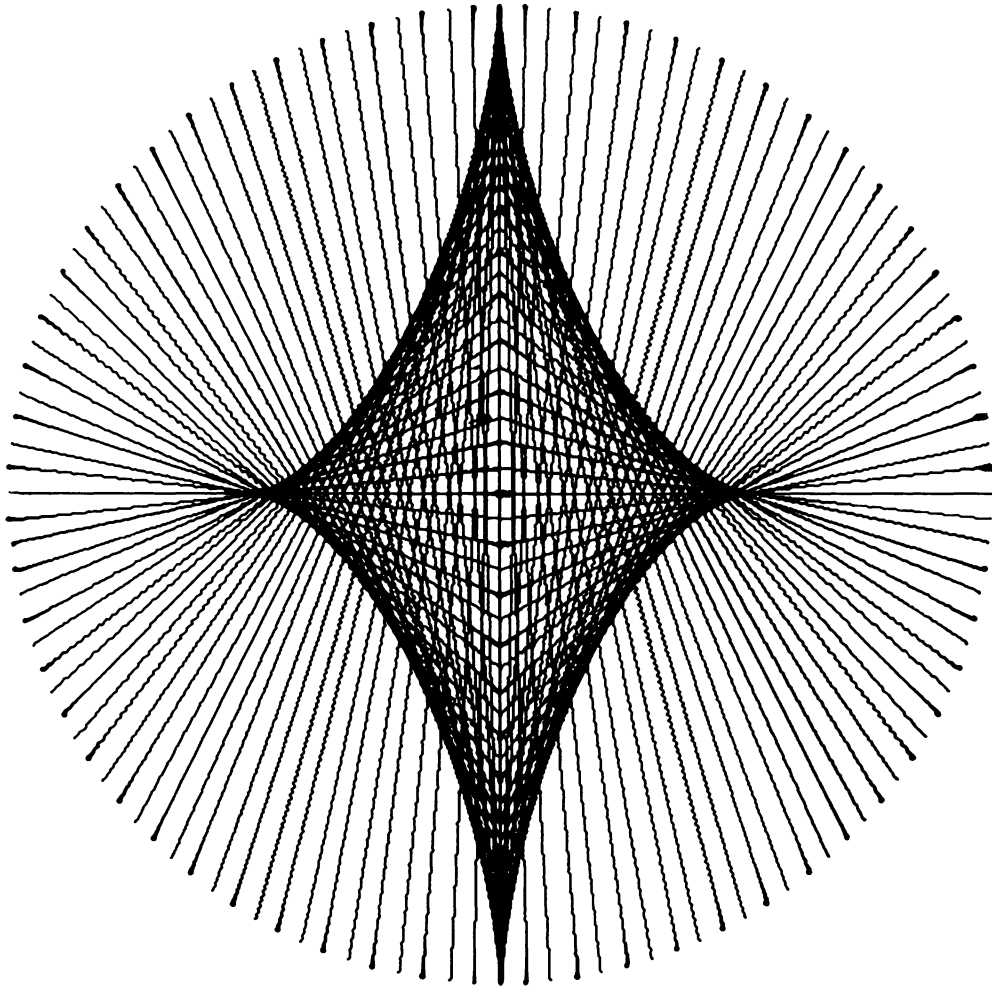


PROGRAM RAY PLOT

Figure 1



PROGRAM KREICH PLOT
Figure 2



PROGRAM LAMP PLOT

Figure 3

 DIAGNOSTICS AND MEMORY MAP

DIAGNOSTICS

Diagnostics, prepared by the compiler may be output on cards, paper tape, on-line printer, or magnetic tape. Diagnostics indicate errors in the program language, but not in program logic.

When the first error is encountered during a compilation, the system stops the object program output. It rewinds the binary output tape and writes a coded error diagnostic over the object program. The system continues compiling the program and outputting diagnostics as additional errors are encountered. When the compilation is complete, the system reads the diagnostic information back into the computer, converts it to a meaningful format, and outputs it on the specified unit. If the output unit is the printer, the diagnostics immediately follow the source program listing. When there are no diagnostics, the memory map immediately follows the source program.

Two error messages can occur before the diagnostics listing.

UNASSIGNED LABELS (list of labels)	types of errors that may cause this message: misspelling the name of a library function, using a non-existent statement number.
OBJECT CODE EXCEEDS MEMORY	the system will continue to compile the entire source program, but the object code will be destroyed.

160-A FORTRAN diagnostics have the following format:

IDENT	LEVEL	INCREMENT	ERROR
-------	-------	-----------	-------

IDENT - statement number

LEVEL - sequence number of subprogram containing error. Sequence numbers are assigned in chronological order - first subprogram encountered is level 1.

INCREMENT - number of statements beyond the last identified statement (IDENT). If the error occurs in a numbered statement, INCREMENT will be 0.

ERROR - a message describing the error.

MEMORY MAP

A memory map prepared by the compiler during compilation is output on cards, paper tape, on-line printer, or magnetic tape. Each line of output is limited to a maximum of 80 characters; this format is constant for any output unit.

A memory map is optional and may be suppressed. If the memory map is to be suppressed, the memory map subroutine is not loaded as a part of the system. The system, however, is designed so that if errors are detected during compilation, operation halts to allow the operator to bring the necessary routine into core. Failure to initiate this action will cause the run to be a complete loss. Suppressing memory map output and changing Standard Equipment Table assignments are explained in 160-A FORTRAN Operating Instructions.

A memory map is essentially a symbol table showing the location of the program in memory. In debugging it may be used to locate incorrectly punched variables and constants. If the programmer has access to the console, he can check the contents of a particular location specified in the memory map. A memory map includes only that information needed by a particular program. For instance, if there are no integer arrays in the program, the memory map will not give INTEGER ARRAYS (See Sample Programs for an example).

INTEGER VARIABLE

IDENT	LEVEL	OBJECT CODE	LOCATION	UP	SUBROUTINE
-------	-------	-------------	----------	----	------------

FLOATING POINT VARIABLE

IDENT	LEVEL	OBJECT CODE	LOCATION
-------	-------	-------------	----------

INTEGER ARRAYS

IDENT	LEVEL	OBJECT CODE	LOCATION	DIMENSION	DIM 1	DIM 2
-------	-------	-------------	----------	-----------	-------	-------

FLOATING POINT ARRAYS

IDENT	LEVEL	OBJECT CODE	LOCATION	DIMENSION	DIM 1	DIM 2
-------	-------	-------------	----------	-----------	-------	-------

CONSTANTS

VALUE	OBJECT CODE	LOCATION
-------	-------------	----------

SUBPROGRAMS

IDENT	LEVEL	OBJECT CODE	LOCATION
-------	-------	-------------	----------

INTEGER VARIABLE USED AS SUBPROGRAM ARGUMENTS

IDENT	LEVEL	ERASABLE	LOCATION	UP	SUBROUTINES
-------	-------	----------	----------	----	-------------

INTEGER ARRAYS USED AS SUBPROGRAM ARGUMENTS

IDENT LEVEL ERASABLE LOCATION DIMENSION DIM 1 DIM 2

FLOATING POINT VARIABLE USED AS SUBPROGRAMS ARGUMENTS

IDENT LEVEL ERASABLE LOCATION

FLOATING POINT ARRAYS USED AS SUBPROGRAMS ARGUMENTS

IDENT LEVEL ERASABLE LOCATION DIMENSION DIM 1 DIM 2

STATEMENT NUMBERS

IDENT LEVEL OBJECT CODE LOCATION

FORMAT STATEMENT

IDENT LEVEL OBJECT CODE LOCATION

LIBRARY FUNCTIONS

IDENT LEVEL OBJECT CODE LOCATION

ERASABLE STORAGE

- IDENT - identification (name) of the variable, array, subprogram, etc.
- LEVEL - the number of the (sub)program containing that particular item. Number assignment is related to the sequence of the (sub)programs except when IDENT is IO and LEVEL is zero. This indicates array input/output by name and is present in all programs.
- OBJECT CODE LOCATION - first gives the bank designator, then the address of the location containing the object code or in the case of arrays, subprograms, and library functions, the first address.
- UP SUBROUTINE - starting address and bank designator of a routine to increment an index or set of indexes. If there is no incrementing, NOT USED is printed.
- DIMENSION - number of subscripts for the array.
- DIM 1 - if the array has 2 subscripts, the critical (first) one is listed.
- DIM 2 - if the array has 3 subscripts, the critical (first two) are listed.
- VALUE - the value of the constant. If the value is outside the size limits for constants, EXPONENT EXCEEDS 32 will be printed under CONSTANTS (normal version only).

ERASABLE LOCATION - is used to determine the address containing a subprogram argument. The bank designator is given; the number following it indicates how far past the first erasable storage address the location is. The first location of erasable storage corresponds to erasable location 0000 (which contains the return address for the subprogram). Each number in ERASABLE LOCATION represents three 160-A words.

Example:

```
ERASABLE STORAGE 1 0060 to 1 1060
ERASABLE LOCATION 1 0002 (bank designator is 1)
```

Addresses	0060	
	0061	Erasable Location
	0062	0000
	0063	
	0064	Erasable Location
	0065	0001
	0066	
	0067	Erasable Location
	0068	0002

For more than one subprogram, the ERASABLE LOCATION indicator is incremented to avoid overlap.

ERASABLE STORAGE - defines the area in memory reserved for ERASABLE LOCATIONS. This area is located between the object program and the data. The left number is the first address after the last object code location; the right address is one less than the last data address; the data is stored in the opposite direction.

FORTRAN 160-A DIAGNOSTICS

ALGEBRAIC EXPRESSION LEFT OF =	MISPLACED COMMON OR EQUIVALENCE
COMPILER TRANSLATION ERROR	MISSING DIMENSION PARENTHESIS
CONVERTED NUMBER IS TOO LARGE	MISSING FORMAT PARENTHESIS
DATA STORAGE EXCEEDS MEMORY	MISSING SENSE SWITCH NUMBER
DUPLICATED FORMAT STATEMENT	MISSING) IN IF STATEMENT
ERROR IN COMMON STATEMENT	MISSING) IN CALL STATEMENT
ERROR IN EQUIVALENCE STATEMENT	MISSING , IN IF STATEMENT
ERROR IN FORM OF PAUSE OR STOP	MORE SUBSCRIPTS THAN DIMENSIONED
ERROR IN LABEL IN DO STATEMENT	MORE THAN THREE DIMENSION
ERROR IN SUBSCRIPT EXPRESSION	MUST HAVE NUMERIC DIMENSION
FLOATING NAME IN FORMAT LABEL	NO END BEFORE SUBROUTINE
ILLEGAL BCD CHARACTER READ	NO FORMAL STATEMENT LABEL
ILLEGAL CHARACTER IN NUMBER	NON MATCHING PARENTHESIS
ILLEGAL EXPONENTIATION	NO OPERAND AFTER OPERATOR
IMPROPER ARRAY NAME	NO OPERAND BETWEEN OPERATORS
IMPROPER CHARACTER IN I/O LIST	NO OPERATOR BETWEEN OPERANDS
IMPROPER CHARACTER IN STATEMENT	ORIGINAL COMMON AREA EXCEEDED
IMPROPER DO NESTING	PREVIOUS ASSIGNMENT OF LABEL
IMPROPER OCTAL NUMBER	PROBABLE MACHINE ERROR
IMPROPER STATEMENT LABEL	PROBABLY IMPLICIT MULTIPLICATION
IMPROPER STATEMENT NUMBER	SHOULD BE COMMA OR RIGHT PAREN
IMPROPER MAGNETIC TAPE LABEL	SIMPLIFY ALGEBRAIC EXPRESSION
INCORRECT SUBROUTINE FORMAT	STATEMENT TOO LONG TO PROCESS
IO-LIST OR DO-LOOP CONTROL ERROR	STATEMENT TYPE NOT IMPLEMENTED
LEADING OPERATOR (NOT + OR -)	TOO MANY CHARACTERS IN NAME
MACHINE ERROR OF UNKNOWN TYPE	VARIABLE NAME ALREADY ASSIGNED

APPENDIX

C

BCD - FLEXOWRITER - TYPEWRITER EQUIVALENCE CODES

Character Equivalence Table

<u>Character</u>	<u>BCD Card Code</u>	<u>Flexowriter</u>		<u>Typewriter</u>	
		<u>UC</u>	<u>LC</u>	<u>UC</u>	<u>LC</u>
1	01		74		74
2	02		70		70
3	03		64		64
4	04		62		62
5	05		66		66
6	06		72		72
7	07		60		60
8	10		33		33
9	11		37		37
0	12		56		56
blank	20	04	04	04	04
dash -	14	52	52	52	52
minus -	40	52	52	52	52
slash /	21		44		44
A	61	30		30	
B	62	23		23	
C	63	16		16	
D	64	22		22	
E	65	20		20	
F	66	26		26	
G	67	13		13	
H	70	05		05	
I	71	14		14	
J	41	32		32	
K	42	36		36	
L	43	11		11	
M	44	07		07	
N	45	06		06	

<u>Character</u>	<u>BCD Card Code</u>	<u>Flexowriter</u>		<u>Typewriter</u>	
		<u>UC</u>	<u>LC</u>	<u>UC</u>	<u>LC</u>
O	46	03		03	
P	47	15		15	
Q	50	35		35	
R	51	12		12	
S	22	24		24	
T	23	01		01	
U	24	34		34	
V	25	17		17	
W	26	31		31	
X	27	27		27	
Y	30	25		25	
Z	31	21		21	
=	13	42			02
+	60	46			46
.	73		42	42	42
)	74		54	56	
\$	53	50		62	
*	54	44		74	
,	33		46	40	40
(34	54		37	

Special Function Codes

<u>Character</u>	<u>BCD Card Code</u>	<u>Flexowriter</u>	<u>Typewriter</u>
Carriage Return		45	45
Lower Case Shift		57	57 (Output Only)
Upper Case Shift		47	47 (Output Only)
Apostrophe' or quotation mark'' changes case code for typewriter input			54 (UC or LC)
Tab is a delete record code for typewriter input			51

GLOSSARY

ARRAY	A group of consecutive storage locations reserved for data storage. Each element of the array is referenced by the array name plus a subscript that indicates its position in the array. 160-A FORTRAN allows one, two, and three dimensional arrays.
BCD	An abbreviation for binary coded decimal, a six-bit binary notation for representing alphabetic, numeric, and special characters. (See Appendix B.)
BINARY	Characterized by two, as a binary operator, such as +, which operates on two quantities, or the binary number system, which has only two digits, 0 and 1.
BIT	An abbreviation for binary digit--a digit which can be 0 or 1.
BOOLEAN	See Masking.
CALLING PROCEDURE	The statements used to transfer control to a subroutine. The calling procedure may transmit variables to the subroutine and return values to the main program from the subroutine.
COMPILER	A computer program which translates non-machine language source programs into machine language object programs, generally producing more than one machine instruction for each source program statement. The 160-A FORTRAN compiler translates 160-A FORTRAN programs into 160-A machine language programs.
DATA LIST	A list of variables for input or output.
DIAGNOSTICS	Statements written on the FORTRAN program listing by the compiler, indicating errors detected in the source program.
EXECUTABLE STATEMENT	A statement which initiates a computer operation in the object program. Non-executable statements are used by the compiler to reserve memory locations and to set initial conditions for a program.
FIXED POINT	A notation for numbers in which the decimal point or binary point is explicitly stated without modification by a scale factor. In FORTRAN all fixed point numbers are integers, that is, the decimal point is not written and is assumed to be immediately to the right of the least significant digit.

FLEXOWRITER	A brand of electric typewriter which has paper tape reading and punching mechanisms.
FLOATING POINT	A notation for numbers in which both a decimal point and a scaling factor are used. The scaling factor indicates a power of 10 to be used as a multiplier of the number.
HOLLERITH	A coding scheme for representing letters, digits, and special characters either by holes punched in cards or paper tape, or by binary numbers.
LEFT-JUSTIFY	To place the first character of a quantity in the left most position in the area in which it is contained.
INTEGER	A whole number without a fractional part.
MASKING	Logical operations used to select parts of constants for numerical operations.
MEMORY MAP	A listing of memory locations reserved by the compiler for the program.
OBJECT LANGUAGE	The programming language produced by a compiling or translating process. The 160-A FORTRAN compiler produces a 160-A machine language object program from a 160-A FORTRAN source program.
OCTAL	An eight-digit (0-7) number system in which the digit positions represent powers of eight. The octal equivalent of a number represented by binary digits can be found by converting each successive group of three binary digits to its integer value.
RIGHT-JUSTIFY	To place the last character of a quantity in the right most position in the area in which it is contained.
SOURCE LANGUAGE	The programming language used in writing a program.
SPECIFICATION LIST	A list of conversion, heading, and spacing specifications occurring in a FORMAT statement.
STORAGE LOCATION	A location, or word, in the computer memory. Each 160-A memory location contains 12 binary digits. Each FORTRAN 160-A integer is stored in two consecutive memory locations and each floating point constant is stored in three consecutive memory locations. When operations involving variables are specified, the computer decides on the basis of the mode of the variable whether to use two or three memory locations.
UNARY	Characterized by one, as a unary operator such as negation or complementation which acts on a single operand.
WORD	See Storage location.

MAGNETIC TAPE

BINARY MODE

Only binary tapes prepared by 160-A FORTRAN can be read in by the system. A binary record contains 121 twelve-bit words or less. If more than 121 words are output, enough 121 word records will be written to include all the data. The total output is referred to as one logical record and the separate 121-word records as physical records.

In 160-A FORTRAN, each physical record contains 121 twelve-bit words. The first word of each physical record (except the last) is a code word containing all zeros. In each logical record, the identifier word of the last physical record contains an 8-bit code followed by four binary ones. The 8-bit code is the binary representation of the number of physical records contained in the logical record.

A logical record in 160-A FORTRAN can be composed of any amount of information up to a limit of 15,488 words. The logical record limit applies to any 160-A system, regardless of the number of banks used by the computer. A BACKSPACE statement will always cause a backspace over an entire logical record. Similarly, a READ TAPE statement will read an entire logical record (minus the code words); or, if enough memory space is not specified in the data list, it will read in the specified variables and then skip to the end of the logical record.

When writing a tape in binary mode, each integer or floating point quantity requires three 12-bit words on the tape. The contents of the third tape word of an integer quantity are meaningless. Binary tapes are written in odd parity.

CODED MODE

All input/output statements imply coded (BCD) mode except for the READ TAPE and WRITE TAPE statements. Input/output statements implying coded mode require that a FORMAT statement designator be named. When reading BCD tape, the number of characters per record varies according to the function of the input tape. If a source program is being read, the system reads either to an end-of-record gap, or through 72 characters; a record is terminated by either condition. Excess characters in a record will be lost. A maximum of 121 data characters per record may be read or written under FORMAT control.

When writing BCD records, many tape to printer routines treat the first character of a 121-character record as a printer control character. This character is not printed. A record of less than 120 characters may be written. An attempt to create a record which exceeds the maximum will result in the loss of all characters after the 121st.

BCD tapes are written in even parity.

PROCEDURES

The following discussions refer to binary and BCD tapes; internal refers to reading the system tape and reading and writing the pre-object code on the scratch tape. Compiler refers to source input or memory map output and object refers to the executable subroutines in the library.

Accumulator Codes for Tape Operation Halts

<u>Mode Number</u>	<u>Tape Set Number</u>	<u>Rewind Code</u>	<u>Logical Tape Number</u>
1 = BCD operation	1 = first 4-tape set	6 = rewind code	1 - 4 = tape number on designated cabinet
2 = binary operation	2 = second 4-tape set		

READ OPERATIONS

When an end-of-file is encountered:

Internal	The A register is set to non-zero on return to the main compiler.
Compiler	The EOF is ignored and the next record is read.
Object	Input buffer is set to blank; low core flag is set for testing by an XEOF function.

When an end-of-tape is encountered:

Compiler and Object	The tape is rewound to load point. Operation halts with 216X or 226X in the accumulator for binary mode, 116X or 126X in accumulator for coded mode. A register will be non-zero. If A is still non-zero when the run switch is set the record in the buffer will be used. If A is zero, a new record will be read. This procedure allows for mounting a new tape.
---------------------	--

If a parity error occurs:

- Internal The system tries to re-read the record three times. If the error persists, the operation halts. The faulty record can be ignored by setting the Run/Step switch on RUN.
- Compiler On a 1607 tape drive, the system tries to re-read the record three times then halts. On a 163 tape drive or equivalent, the tape is backspaced three records, repositioned, and the current record is re-read three more times before halting. The record will be ignored if RUN is set.
- Object The operation is the same as for compiler except the record will always end up in even parity (BCD).

WRITE OPERATIONS

The system does not recognize an end-of-file encountered during the write process.

When an end-of-tape is encountered:

- Compiler The system backspaces, writes two end-of-files, rewinds and the tape and halts the operation. If the equipment configuration includes no more than four tapes, the accumulator will contain 216X; X is the logical tape number. If the tape is located in a second set of four tapes, the accumulator will contain 226X. Setting the Run/Step switch to RUN will cause the tape to be written again.
- Object

If a parity error occurs:

- Internal On a 163 or equivalent, the system backspaces the tape one record and tries to re-write the record (three times). If the parity error persists, the system writes an end-of-file mark, skips 6 inches of tape and tries three more times to write the record. This moving and writing process will be repeated two more times if necessary.

On the 1607, the system will backspace the tape and try to re-write the record only once before skipping tape.

In both cases, if the run switch is set, the record will be ignored and the WRITE operation is completed.

STATUS CHECKING On a 163 or equivalent tape drive, the only check is made after the READ or WRITE; there is no wait until ready loop.

On the 1607, status is checked for ready before any operation is performed as well as after any READ or WRITE.

PAPER TAPE AND TYPEWRITER

PUNCHED PAPER TAPE

Punched paper tape can be used for input/output in the 160-A FORTRAN system. A Flexowriter is used to prepare and list paper tape output. Source programs and data can be read from paper tape; an object program memory map, program diagnostics, and results can be punched on paper tape for listing on the Flexowriter.

The 160-A FORTRAN system recognizes only legal flex characters for either Flexowriter input or output. Illegal flex characters are converted to blanks during processing. If an illegal character occurs in a numeric data field, the blank is further converted to zero. A Flexowriter tab is an illegal character for data input. A carriage return should not be used as the first character in a source program or as the first character in data input since the carriage return is interpreted as an end-of-record.

When the Flexowriter is used for input/output, a line constitutes a record, and a record contains a maximum of 120 characters. Delete characters, carriage returns, and case codes do not count as spaces in a record.

Case codes and several other paper tape characters which affect the translation process are not included in the FORTRAN character set. Flexowriter case codes which appear as characters on paper tape have a mechanical function only. The 160-A FORTRAN system automatically sets up a lower case condition at the beginning of each record. When necessary, upper case must be re-specified at the beginning of a record even though the previous record ended in upper case. Flexowriter operators preparing 160-A FORTRAN input must be informed that an upper case code condition applies only to the line in which it appears. They also must be instructed to punch an apostrophe (') for an asterisk (*) and a colon (:) for a dollar sign (\$). The \$ sign is legal only in a Hollerith field. Flexowriter codes and the spaces they occupy within the computer need not be taken into consideration when planning input/output under program control; system input/output routines handle such details automatically.

TYPEWRITER

The typewriter can be used only for data input or output. Mechanical restrictions affect the use of upper and lower case shifts during input. Consequently, users must ascertain that typed input meets all requirements.

INPUT

Each time the system is ready to accept an input record from the typewriter, it returns the carriage, types a question mark, and sets the typewriter to lower case. A data record is a typed line, 120-characters maximum; each record is terminated by a carriage return. Only legal BCD characters should be used since illegal characters are converted to blanks. If the illegal characters occur in a numeric data field, blanks are further converted to zeros.

An error may be corrected by striking the typewriter tab bar. This repositions the typewriter to column one, erases the previous record, and resets the typewriter to accept input.

Because case code cannot be input from the 160-A typewriter, the 160-A FORTRAN system uses a pseudo change of case code. The system recognizes an apostrophe (') or a quotation mark (") as a signal to set the case code opposite to the one currently in effect. If a record is begun in normal lower case, striking an apostrophe or quote mark shifts the typewriter into upper case. This upper case will continue until the end of the record, or until another apostrophe or quote mark is used. There is no limit on the number of case shifts which can be made in a single record. Apostrophes and quote marks do not count as characters in a consideration of record length.

OUTPUT

When the typewriter is specified as the output medium, a carriage return precedes each data record. Any legal BCD character can be typed out. Special typewriter symbols not recognized as legal BCD characters are treated as illegal and are converted to blanks.

Output is single spaced and all alphabetic characters are written in upper case.

STATEMENT INDEX

X = executable

N = nonexecutable

Statement	Type	Page
ASSIGN	X	22
BACKSPACE i	X	60
CALL	X	33
COMMON	N	38
COMMON (n)	N	40
CONTINUE	X	28
DIMENSION	N	37
DO	X	24
END	*	28
ENDFILE	X	60
EQUIVALENCE	N	41
FORMAT	N	48
GOTO i	X	22
GOTO n	X	21
GOTO() i	X	22
IF ACCUMULATOR OVERFLOW	X	27
IF(E)n1,n2,n3	X	23
IF DIVIDE CHECK	X	27
IF QUOTIENT OVERFLOW	X	27
IF(SENSE SWITCH i)	X	23
PAUSE	X	28
PRINT	X	62
PUNCH	X	61

*END is executable only under certain circumstances. See 5.2.3 as well as 4.9.

Statement	Type	Page
PUNCH FLEX	X	61
READ	X	60
READ FLEX	X	61
READ INPUT TAPE	X	60
READ TAPE	X	60
READ TYPE	X	61
RETURN	X	33
REWIND	X	60
STOP	X	28
SUBROUTINE	N	31
WRITE OUTPUT TAPE	X	60
WRITE TAPE	X	60
WRITE TYPE	X	61

TOPIC INDEX

	Page
Arithmetic Expressions	11
computed IF statement	23
mode	14
operators	11
ordering of operations	12
subscripts (use as)	18
Arithmetic statements	19
Arrays	7
single subscript form	8
storage	8
subscripts	18
used in COMMON statement	38
data lists	45
DIMENSION statement	37
Blanks	
used in constants	5,6
data fields	56
formats	56
variable names	7
Boolean — see Masking	
Coding form	63
Comments	64
Constants in source program	5
floating point	5
integer	5
masking	6

	Page
Constants in input data — see Data conversion	
Continuation	64
Control statements	21
numerical COMMON statement	40
statement in DO loops	24
subroutines	30
Data conversion	49
alphanumeric — A	55
floating-point — E	51
floating point — F	53
integer — I	49
octal — O	50
Data list — input-output	45
Deck structure	67
COMMON statements	38
DIMENSION statements	37
Diagnostics	93
Divide check	27
DO loop	25
Equivalence	41
Executable statements	104
Exponentiation	14
Fault Tests	27
Flexowriter	103
data input	103
input-output statements	60
source program input	65
Formal parameters	32
Format	48
complete specifications	57
data conversion	49
headings and spacing	56

	Page
repeated specifications	49
reserved words	5
variable format control	58
Functions	29
in DO loop range	24
Headings in formats	56
Identification field	64
Index variables	
data lists	45
DO loops	24
Magnetic tape	99
logical tape unit numbers	59
physical and logical records	99
recording mode	99
source program input	67
statements	59
Masking expressions	15
computed IF statement	22
operators	15
ordering of operations	17
Masking statements	19
Memory map	89
Non-standard input-output equipment	62
Numerical common	40
Operators	
arithmetic	11
masking	15
Overflow tests	27
Plotter	81
Printer	
carriage control	62
statements	62

	Page
Punched cards	64
data input	67
statements	60
source program input	67
Range of Constants	
floating point	6
integer	5
masking	6
Reserved words	5
Sense switches	23
Spaces — see blanks	
Statements	
arithmetic	19
list of 160-A statements FORTRAN	104
masking	19
replacement	18
Statement identifiers	21
Storage allocation	8,37,40
COMMON	38
overlay	39
Subroutines	30
calling	33
in DO loops	24
linkage — COMMON	38
linkage — formal parameters	32
Subscripts	
array elements	18
in DIMENSION statement	37
legal types	18
reduction to single subscript	8
Typewriter	103

	Page
data input	103
statements	61
Variables	7
index	24,48
in masking statements	19
simple floating-point	7
simple integer	7
subscripted floating point	9
subscripted integer	9

CONTROL DATA SALES OFFICES

ALAMOGORDO • ALBUQUERQUE • ATLANTA • BILLINGS • BOSTON • CAPE
CANAVERAL • CHICAGO • CINCINNATI • CLEVELAND • COLORADO SPRINGS
DALLAS • DAYTON • DENVER • DETROIT • DOWNEY, CALIFORNIA • HONOLULU
HOUSTON • HUNTSVILLE • ITHACA • KANSAS CITY, KANSAS • LOS ANGELES
MADISON, WISCONSIN • MINNEAPOLIS • NEWARK • NEW ORLEANS • NEW
YORK CITY • OAKLAND • OMAHA • PALO ALTO • PHILADELPHIA • PHOENIX
PITTSBURGH • SACRAMENTO • SALT LAKE CITY • SAN BERNARDINO • SAN
DIEGO • SEATTLE • WASHINGTON, D.C.

ATHENS • CANBERRA • FRANKFURT • THE HAGUE • HAMBURG • JOHA
BURG • LONDON • MELBOURNE • MEXICO CITY (REGAL ELECTRONI
MEXICO, S.A.) • MILAN • MUNICH • OSLO • OTTAWA (COMPUTING DI
OF CANADA, LIMITED) • PARIS • STOCKHOLM • STUTTGART • S
TOKYO (C. ITOH ELECTRONIC COMPUTING SERVICE CO., LTD.) • Z

CONTROL DATA
CORPORATION

8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440