CONTROL DATA CORPORATION

CONTROL DATA®
1700 COMPUTER SYSTEMS

1700 MSOS 4
MS FORTRAN VERSION 3A/B
GENERAL INFORMATION MANUAL

REVISION RECORD

REVISION	DESCRIPTION
A	Manual released.
(7/74)	
Publication No. 39519900	

Address comments concerning this manual to:

Publications & Graphics Division
4455 Eastgate Mall
La Jolla, California 92037

or use Comment Sheet in the back of this manual.

CONTENTS

1700 FORTRAN SYSTEM 1

The 1700 Mass Storage FORTRAN system for the Control Data® 1700 computer provides a convenient language for expressing mathematical and scientific problems in a familiar mathematical notation.

A set of FORTRAN statements to accomplish a particular task is accepted as a source program by the FORTRAN compiler; the object program produced by the compiler contains the machine language commands to solve the problem. Object programs may be run repeatedly with varying sets of data.

1700 Mass Storage FORTRAN is ASA FORTRAN with the differences described in the following text. Many programs written in ASA FORTRAN can be compiled by 1700 FORTRAN with little modification. All Basic FORTRAN programs can be compiled correctly by 1700 FORTRAN.

The 1700 Mass Storage FORTRAN source language includes the following features:

- Constants and variables of types:

Integer	Real (floating-point)
Hexadecimal	Double precision
Byte	Single
Signed Byte	ANSI

- Library functions
- Independently compilable subprograms
- Generalized subscript expressions
- Variable format for input/output control
- Bit and byte manipulations
- Run-anywhere compile time option
- Mass storage language statements
- Double-precision floating-point package

The 1700 Mass Storage FORTRAN Version 3 product is composed of five basic elements:

- A Variant FORTRAN Compiler — This compiler version has a larger number of overlays; the largest overlay is approximately 8K. It requires more mass memory than the B variant and is slower in compilation speed.

- B Variant FORTRAN Compiler — This compiler has fewer overlays than the A variant; the largest overlay is approximately 16K. This variant is faster than the A variant. Both compilers process source statements identically and generate identical object code.
- Re-entrant ENCODE/DECODE Run-Time — This run-time runs in the foreground and has the characteristics for multiprogramming.
- Non-Re-entrant ENCODE/DECODE Run-Time — This run-time runs in the background and has identical user interface as the Re-entrant ENCODE/DECODE run-time. This run-time is designed for use in debugging programs to obtain the foreground.
- FORTRAN I/O Run-Time — This run-time runs in the background and has more extensive capabilities than the other two run-times.

PRODUCT CONFIGURATIONS

Several configurations are possible using the five elements of the product.

Only one variant of the compiler may be present in a given MSOS system. With the selected compiler, Re-entrant ENCODE/DECODE Run-Time may be used (must be core-resident). Either Non-Re-entrant ENCODE/DECODE Run-Time or FORTRAN I/O Run-Time may be in the background. In addition, if FORTRAN I/O Run-Time is in the background, the non-duplicative functions present in Non-Re-entrant ENCODE/DECODE can also be in the background.

Specific details of the configurations can be found in the MSOS Configuration Manual, CDC Publication No.

PRODUCT HARDWARE REQUIREMENTS

The MSOS Reference Manual should be consulted for specific hardware options available (CDC Publication No. 60361500).

The minimum system memory requirements for MSOS do not include any of the elements of Mass Storage FORTRAN. If the A variant of the compiler is used, the minimum memory requirement is 24K. The B variant minimum is 32K. If the foreground ENCODE/DECODE run-time is used, the additional memory requirement is 4K for single-precision floating-point, or 8K for double-precision floating-point.

OUTPUT

Output selected by the programmer may include:

- Relocatable object program

- Source program listing plus diagnostics
- Object program listing (binary and assembly code equivalent)
- Load-and-go object program for immediate execution

Diagnostic messages are printed when the compiler detects actual errors and probable errors.

COMPILER OPTIMIZATIONS

1700 Mass Storage FORTRAN is a multiple-pass compiler which produces highly optimized code. The optimizations are listed below:

- Common subexpressions, including subscripts, within or between arithmetic expressions are identified and computed only once.
- Subexpressions are computed at the lowest DO loop level.
- Subscripts being acted upon by DO loop induction variables are computed recursively.
- Index registers are optimally assigned.
- One word relative addressing is used where possible.
- Storage is allocated to maximize relative addressing.
- All simple FORTRAN-provided functions are inserted in-line; for example, IABS or AND.
- A comprehensive analysis of IF statements is made. A transfer from the IF statement to the label of the next statement is recognized in the generated code. In a logical IF, the computations are structured to determine the truth value with the least computation.
- The analysis and computation of arithmetic expressions are accomplished in an order which minimizes both the amount of code generated and the execution time.
- Division by a real constant is accomplished through multiplication by the reciprocal of the constant.
- For integer variables, multiplication and division by a constant which is a power of two is accomplished through shifting. When numbers are raised to integer constant powers, in-line multiplication is used wherever it increases efficiency.
- The values in the A, Q, and I registers are retained and may be used later.
- A flow analysis of the program is made; common subexpressions and index register assignments are carried through the flow.

CARD FORMATS

The initial statement card format is:

Columns	Content
1-5	Blank or statement label
6	Blank or 0
7-72	Statements
73-80	Identification and sequencing

The continuation card format is:

Columns	Content
1-5	Blank
6	Any character other than 0 or blank
7-72	Continuation statement
73-80	Identification and sequencing

Up to five continuation cards are allowed for a statement.

The comment card format is:

Columns	Content
1	C -- comment designator
2-72	Comments
73-80	Identification and sequencing

SOURCE PROGRAMS

A source program may be a main program or a subprogram. Source programs must be compiled separately but may be run together. All specification statements must be placed at the beginning of the source program.

Data values may be entered by DATA declarations. Specific storage areas may be reserved by COMMON statements for reference by subprograms and the associated main program. EQUIVALENCE allows the programmer to overlay the same storage locations with variables and arrays during program execution.

The mode of a variable, integer or real, may be defined by a type declaration or by the form of the variable itself.

Arithmetic operations include: addition, subtraction, multiplication, division, and exponentiation. Logical statements may include relational and logical operators.

Control statements may alter the sequential execution of instructions unconditionally or dependent upon the value of an expression.

Input/output operations transmit data between the computer storage and external equipment. Conversion and editing specifications permit great diversity in input/output formats.

ELEMENTS

CHARACTER SET

Alphabetic:	A through Z	
Arabic numerals:	0 through 9	
Special characters:		Blank
	=	Equal sign
	+	Plus
	-	Minus
	*	Asterisk
	/	Slash
	(	Left parenthesis
	)	Right parenthesis
	,	Comma
	.	Decimal point
	$	Currency symbol
	'	Apostrophe or single quote
	!	Exclamation point
	"	Quotation marks
	#	Number sign
	%	Percent sign
	&	Ampersand
	:	Colon
	;	Semicolon
	<	Less than
	>	Greater than
	?	Question mark
	@	Commercial at
	[	Opening bracket
	\	Reverse slant
	]	Closing bracket
	^	Circumflex
	_	Underline

CONSTANTS

Integer, real, hexadecimal, and ANSI constants are variable. Each type has a different mathematical significance and a different internal representation. The type of constant is determined by the form in which it is written or by the context.

INTEGER

Integer constants are always exact representations of integer values with a range in magnitude of from 0 to $2^{15}-1$. They may assume positive and negative values.

Integer constants may be represented in decimal form (0-9) or hexadecimal form (0-9, A-F). When hexadecimal representation is used, the integer value must be preceded by a dollar sign ($).

REAL

Real constants are approximations of real numbers with a range in magnitude of from 0 to 2^{128}. They may assume positive and negative values. Significance is one part in eight million. Single- and double-precision real capability is provided.

ANSI

The FORTRAN characters and their corresponding American National Standards Institute (ANSI) codes are listed in Appendix B.

VARIABLES

Variables are alphanumeric identifiers which represent specific storage locations. Simple and subscripted variables are recognized. A variable may be designated as a byte of another variable through a BYTE or SIGNED BYTE statement. Such a variable is treated as a signed or unsigned integer when used in the body of the program.

When a variable is not declared by a type statement, it is assumed integer if the initial character is I, J, K, L, M, or N. If the variable begins with another alphabetic character, it is assumed real.

ARRAY

An array represents a block of successive storage locations for variables. Each element of the array is referenced by the array name plus 1, 2, or 3 subscripts in the following forms (c and d are unsigned integer constants and m is a simple integer or byte variable):

d	c * m
m	c * m ± d
m ± d	

A reference to an array must contain the number of subscripts specified in the DIMENSION statement.

Examples of subscripted variables:

```
A (I,J)
C (14)
BA (J + 3,5)
Q (I-1,J,2*K)
```

RUN-ANYWHERE OPTION

Selection of this option results in an object program which will run correctly anywhere in allocatable core, independent of the location at which it is loaded.

ASA OPTION

This option provides ASA compatibility. When this option is selected, two words of storage are allocated for integers, only one of which is used. (See SINGLE statement.)

REPLACEMENT STATEMENTS 2

REPLACEMENT STATEMENT

r = e

The value of the expression, e, is assigned to the variable identifier, r; e is an arithmetic expression.

ARITHMETIC EXPRESSION

Any simple or subscripted variable, constant, or function may be an arithmetic expression. These entities may be combined by using the following arithmetic operators to form other arithmetic expressions:

**	exponentiation	+	addition
*	multiplication	-	subtraction
/	division		

RELATIONAL EXPRESSION

e_1 op e_2

A relational expression is true if the arithmetic expressions e_1 and e_2 satisfy the relation specified by the operator, op; otherwise the relation is false. e_1 and e_2 must be of the same type.

Relational operators:

.EQ.	equal to
.NE.	not equal to
.GT.	greater than
.GE.	greater than or equal to
.LT.	less than
.LE.	less than or equal to

LOGICAL EXPRESSION

e_1 op e_2 op ... op e_n

A logical expression is formed with logical operators and logical elements (e_i), and is either true or false.

Logical operators:

.OR.	logical disjunction
.AND.	logical conjunction
Logical primary	A relational expression
Logical factor	A logical primary or .NOT. followed by a logical primary where .NOT. is logical negation
Logical element	A logical primary or a construct of one of the following two forms enclosed in parentheses: 1) logical primary .AND. logical factor 2) logical primary .OR. logical factor

SPECIFICATION STATEMENTS 3

DIMENSION

DIMENSION $v_1, v_2, \ldots, v_n$

Storage locations are reserved for the array identifiers, v_i, which may be subscripted with up to three unsigned integers. The number of locations reserved is computed from the DIMENSION statement and type of array. The arrays will not necessarily be assigned consecutive blocks of storage.

BYTE

BYTE $(x_1, y_1 (c_1 = d_1)), \ldots, (x_n, y_n (c_n = d_n))$

Where:	x_i	is bits c_i through d_i of y_i
	y_i	is an integer variable, integer array, or an integer array element
	c_i and d_i	are integer constants in the range: $15 \geq c_1 \geq d_1 \geq 0$

A variable may be designated as a byte of another variable or array with this statement. Such a variable is treated as an unsigned integer when used in the body of the program.

SIGNED BYTE

SIGNED BYTE $(x_1, y_1 (c_1 = d_1)), \ldots, (x_n, y_n (c_n = d_n))$

Same as BYTE except bytes are treated as signed integers only.

COMMON

COMMON/name/$v_1, v_2, \ldots, v_n$

Where:	name	identifies a common block (blank for blank common)
	v_i	is a variable name, array name, or subscripted array identifier

Common locations are assigned to the identifiers for reference by independently compiled programs and subprograms. Values in labeled common (name is not blank) may be preset by a BLOCK DATA subprogram. Dimensioning information may be supplied.

One block of labeled common and one block of blank common may be declared for a program.

TYPE

type $v_1, v_2, \ldots, v_n$

Where:	type	is INTEGER, REAL, or DOUBLE PRECISION
	v_i	is a variable name, array name, or function name

This statement declares the type of the identifier. It overrides or confirms the type implied by the first character of the identifier and may supply dimension information.

EQUIVALENCE

EQUIVALENCE $(a_1, b_1, \ldots), (a_2, b_2, \ldots), \ldots$

Storage may be shared by two or more entities. The names $a_i, b_i, \ldots$ may be variable names or array element names.

No more than one element in an EQUIVALENCE group may appear in a COMMON statement. No element may be a formal parameter.

DATA

DATA $v_1/d_1/, v_2/d_2/, \ldots, v_n/d_n/$

Where:	v_i	is a list containing names of variables, arrays, array elements, and implied DO loops
	d_i	is a constant, signed or unsigned, or Hollerith text

This statement defines initial values of variables or array elements. A one-to-one correspondence must exist between the list items and the constants. A constant may be preceded by k* to indicate that the constant is to be specified k times. Apostrophes may be used to enclose a constant. In this case, the ANSI code for the symbols in the constant are stored into the corresponding variable.

EXTERNAL

EXTERNAL $name_1, name_2, \ldots, name_n$

This statement specifies the parameter names to be external procedure names.

RELATIVE

RELATIVE $name_1, name_2, \ldots, name_n$

This statement specifies the parameter names to be external procedure names. When the run-anywhere option is selected, all references to this procedure will be made relative. Relative externals may not be passed as parameters to subprograms.

SINGLE

SINGLE $v_1, v_2, \ldots, v_n$

Specifies variables v_i as one-word integers. Dimension information may also be specified. This statement is used if the ASA compile time option is selected.

DATA TRANSMISSION STATEMENTS 4

Data may be transferred within the computer and between the computer and peripheral equipment with the following statements.

FORMAT

FORMAT ($spec_1$, $spec_2$, ..., $spec_n$)

Where: $spec_i$ is conversion of editing specification listed below

Editing specifications and BCD conversion specifications may be included in programs. Format specifications can be compiled into a program or read into an array at object time.

Conversion specifications:

rEw.d	Floating-point with exponent
rFw.d	Floating-point without exponent
rDw.d	Double-precision floating-point with exponentiation
rIw or Iw.d	Decimal integer
r$w or Zw	Hexadecimal conversion
rAw	Alphanumeric
rRw	Alphanumeric

Editing specifications:

wX	Intra-line spacing
wH	ANSI heading and labeling
/	Begin new record
* or '	ANSI heading and labeling

FORMATTED READ/WRITE

READ(i,n)l

WRITE(i,n)l

Where:	i	is the logical unit number
	n	is the FORMAT statement specifying how to move data
	l	is the list of variables to be transmitted

These statements transmit physical records, containing up to 120 characters, between the computer and logical unit i according to statement n which may represent one of the following:

- The label of a FORMAT statement
- An array name
- An assign variable or formal parameter which has been assigned the label of a FORMAT statement

UNFORMATTED READ/WRITE

READ(i)l

WRITE(i)l

These statements transmit a binary record to or from logical unit i. One logical record is produced from each READ or WRITE statement. A logical record may consist of several physical records. The parameters have the same meanings as in formatted READ/WRITE. ENCODE/DECODE and other formatting routines aid the programmer in formatting his own data when necessary.

SETBFR

A call to SETBFR provides the re-entrant FORTRAN I/O package with information regarding where to store the program's I/O requests and corresponding data. It is used in conjunction with FORTRAN I/O requests which are in the foreground.

CALL READ/WRITE/FREAD/FWRITE

In addition to the formatted READ, WRITE and the unformatted READ/WRITE, the FORTRAN programmer can perform 1700 MSOS 4 monitor calls to perform read or write requests by use of the FORTRAN run-time package. These FORTRAN calls have the following forms:

CALL READ
CALL WRITE
CALL FREAD
CALL FWRITE

The CALL READ and CALL WRITE statements permit word addressing of mass memory devices as well as sector addressing. (Only sector addressing of mass memory is possible with the FORTRAN READ/WRITE statements.)

The CALL FREAD and CALL FWRITE statements may be used by a background program to transfer binary information to or from mass memory. (The FORTRAN unformatted READ/WRITE statements cannot be used by a background program to access mass memory if the standard FORTRAN library routines are used.)

These calls do not refer to a FORTRAN FORMAT statement, but require that the user do his own formatting. The routines described in the next sections aid the programmer in formatting data.

ENCODE/DECODE CALLS

```
CALL DECODE  (v,n,c,l)
CALL ENCODE  (v,n,c,l)
```

Where:	v	is the starting address
	n	is the FORMAT statement specifying how to move data or array name
	c	is the number of variables to ENCODE/DECODE
	l	is the list of variables to be transmitted

These statements transmit information, under FORMAT specifications, from one area of internal storage to another.

ADDITIONAL FORMATTING ROUTINES

HEXASC	Converts a number to the ANSI characters corresponding to the digits in the hexadecimal form of the number
HEXDEC	Converts a number to the ANSI characters corresponding to the digits in the decimal form of the number
ASCII	Converts ANSI characters to a number, assuming the ANSI characters represent hexadecimal digits

DECHEX	Converts ANSI characters to a number, assuming the ANSI characters represent decimal digits
AFORM	Converts a word containing two ANSI characters to two words each containing a character left-justified blank-filled
RFORM	Converts a word containing two ANSI characters to two words each containing a character right-justified zero-filled
FLOATG	Converts a floating-point number to ANSI characters including the sign, decimal point, and the exponent of the numoer

FORTRAN/MONITOR RUN-TIME PACKAGE

The FORTRAN/Monitor run-time package enables the FORTRAN programmer to make certain monitor requests, obtain monitor parameters, and execute I/O commands. The calls to READ, WRITE, FREAD, and FWRITE, as discussed earlier, are a part of the run-time package. The other FORTRAN monitor requests are as follows:

CALL SCHEDL	Schedules a requested program at a requested priority
CALL TIMER	After a specified time interval, schedules a requested program at a requested priority
CALL RELESE	Returns memory to the core allocator

In addition to the monitor calls, the run-time package provides the FORTRAN programmer with access to the following routines:

LINK	Obtains the value in the Q register for use by the FORTRAN program
DISPAT	Transfers control to the dispatcher
ICLOCK	Obtains the value of the system clock
OUTINS	Performs output via the 1705 Interrupt/Data Channel
INPINS	Performs input via the 1705 Interrupt/Data Channel
ICONCT	Performs a connect to the 1750 DCB terminator and then inputs from a device connected to the 1750
OCONCT	Performs a connect to the 1750 DCB terminator and then inputs from a device connected to the 1750
ENDFILE	ENDFILE lu causes the recording of an endfile record on the unit identified by lu.

REWIND	REWIND lu positions the unit identified by lu at its load point.
BACKSPACE	BACKSPACE lu causes the unit identified by lu to go back to the beginning of the preceding record.

ACCESSING MASS STORAGE FORTRAN FILES

Mass Storage FORTRAN files may be created and accessed by a FORTRAN program. These files are assigned to the scratch area of the mass storage device and are not retained after execution of a job. (They are not to be confused with File Manager files as described in the 1700 MSOS 4 Reference Manual or with permanent files in the program library.)

To create a mass storage file, an OPEN statement must be executed. The OPEN statement has the following form:

```
OPEN    k,i,j,u,x
```

Where:

k	is the name of the file
i	is the number of sectors per record
j	is the maximum number of records in the file
u	is the logical unit to which file is assigned
x	is the starting sector address for the file (optional)

To access the file, alternate forms of the FORTRAN READ/WRITE statements are used. The alternate forms are as follows:

```
READ    (k(n), f)l
WRITE   (k(n), f)l
```

Where:

k	is the name of the mass storage file
n	is the record number
f	is the format specification, which may be the label of a FORMAT statement, an array name, or a variable which has been assigned the label of a FORMAT statement
l	is the list of variables to be transmitted

ASSEM STATEMENTS 5

Assembly language instructions may be inserted in-line in a 1700 FORTRAN program by use of ASSEM statements. The inserted instructions are specified by the FORTRAN programmer in the following forms:

- Hexadecimal constants which may represent code to be executed or actual constants
- References to statement labels within the program
- References to variables within the program
- References to externals declared by the program
- Indirect addressing indicators

ASSEM statements may be used to generate calling sequences to the operating system and to access the core communication region.

CONTROL STATEMENTS 6

The following statements may be used to alter the sequential execution of instructions.

ASSIGN

ASSIGN k TO i

Where:	k	is the statement label
	i	is the integer variable name (assign variable)

A label assignment statement stores the location of a statement label into a variable.

GO TO

UNCONDITIONAL

GO TO n

Control transfers to the statement identified by n.

ASSIGNED

GO TO i or GO TO i, $(k_i, k_2, \ldots, k_m)$

Where:	i	is the integer variable name (assign variable)
	k_i	is the optional statement labels which may be included for the programmer's convenience; they are not used by the compiler.

Before an assigned GO TO statement is executed, the current value of i must be previously assigned by an ASSIGN statement. Control transfers to that assigned location. The i may be assigned in either the program unit of the GO TO or in another program unit where i was passed as an actual parameter or was in COMMON.

COMPUTED

GO TO $(k_1, k_2, \ldots, k_m)$, i

Where	k_i	is the statement label
	i	is the integer variable reference

Control transfers to statement k_i.

DO

DO statements provide repetitive operation and incrementing.

DO n i = m_1, m_2 or DO n i = m_1, m_2, m_3

Where:	n	is the statement number at the end of a sequence of instructions which begins with the DO statement
	i	is a simple integer variable
	m_i	is an integer constant or simple integer variable

The initial value of i is m_1. The value of i is incremented by m_3 each time. The sequence is repeated until i surpasses the value of m_2. If m_3 is omitted, it is assumed to have the value 1. A DO loop may include other DO loops.

IF

IF statements transfer control conditionally depending on the value of an arithmetic or logical expression.

IF (e) k_1, k_2, k_3

Where:	e	is an arithmetic expression
	k_i	is a statement label

Control transfers to k_1 if the value of e is negative, to k_2 if the value is zero, and to k_3 if the value is positive.

IF (l) s

Where:	l	is a logical expression
	s	is any executable statement except a DO or another logical IF statement

If l is false, s is executed as though it were a CONTINUE statement. If l is true, statement s is executed.

CONTINUE

```
CONTINUE
```

This is a no-operation instruction which may be given a statement number for reference. It is frequently used to terminate a DO loop.

PAUSE

```
PAUSE
PAUSE v
```

Where:	v	is an octal number with a maximum value of 77777

The PAUSE statement halts a program temporarily. The word PAUSE and the value of v, if present, are printed on the output comment device. A carriage return entered by the operator resumes execution with the statement immediately following the PAUSE statement.

STOP

```
STOP
STOP v
```

Where:	v	is an octal number with a maximum value of 77777

The STOP statement terminates the execution of a program. The word STOP and the value of v, if present, are printed on the output comment device.

PROGRAM AND SUBPROGRAM STATEMENTS

SUBROUTINE

SUBROUTINE name $(p_1, p_2, \ldots, p_n)$

Where:	name	is the alphanumeric identifier
	p_i	is a formal parameter (optional)

The first statement in a subroutine defines it. A subroutine may return resulting values through formal parameters.

CALL

CALL name $(c_1, c_2, \ldots, c_n)$

Where:	name	is the alphanumeric identifier
	c_i	is an actual parameter

Control transfers from a program or subprogram to subroutine name with actual parameters, c_i, replacing formal parameters, p_i, in the subroutine parameter list. The actual parameters may be variables, array names, array element names, constants, arithmetic expressions, or external subprogram names.

FUNCTIONS

EXTERNAL

FUNCTION name $(p_1, p_2, \ldots, p_n)$

Where:	name	is the alphanumeric identifier
	p_i	is a formal parameter (optional)

This must be the first statement in a function subprogram. A function returns a single value as a result.

STATEMENT

name $(p_1,p_2,\ldots,p_n)$=e

Where:	name	is the alphanumeric identifier
	p_i	is a formal parameter
	e	is an arithmetic expression involving p_i

This statement defines the value of name, which is inserted in the code wherever name is used as an operand in an expression. The expression e may contain references to library functions, other statement functions, or function subprograms.

The statement function name may not appear in a DIMENSION, EQUIVALENCE, or COMMON statement.

REFERENCE

name $(c_1,c_2,\ldots,c_3)$

Where:	name	is an alphanumeric identifier
	c_i	is an actual parameter

When the statement function appears as an operand in an expression, control transfers to the named function. Control returns to the statement containing the function reference and the value returned is associated with the function identifier. A function reference may be used anywhere that a variable identifier may be used.

Actual parameters may be variables, array names, array element names, constants, arithmetic expressions, or external subprogram names.

BLOCK DATA

BLOCK DATA

Block data subprograms are used to enter initial values into elements of labeled common blocks. This special subprogram contains only specification statements. BLOCK DATA must be the first statement in this subprogram.

If an entity of a particular common block is being given an initial value in such a subprogram, a complete set of specification statements for the entire block must be included, even though some of the elements of the block do not appear in DATA statements. Initial values may be entered into more than one block in a single subprogram.

RETURN

RETURN

This statement signals the end of logic flow within a subroutine or function and returns control to the calling program. More than one RETURN statement may appear within a single subroutine or function subprogram. If RETURN is omitted, the END statement serves as a RETURN statement.

END

END

This statement marks the physical end of a program, subroutine, or function.

PREDEFINED FUNCTIONS 8

BASIC EXTERNAL FUNCTIONS

The following functions may be referenced in any program or subprogram.

Function	Definition	Number of Arguments	Symbolic Name	Type of Argument	Type of Function
Exponential	e^a	1	EXP DEXP	Real Double	Real Double
Natural logarithm	$\log_e(a)$	1	ALOG DLOG	Real Double	Real Double
Trigonometric sine	sin (a)	1	SIN DSIN	Real Double	Real Double
Trigonometric cosine	cos (a)	1	COS DCOS	Real Double	Real Double
Hyperbolic tangent	tanh (a)	1	TANH	Real	Real
Square root	$(a)^{1/2}$	1	SQRT DSQRT	Real Double	Real Double
Arctangent	arctan (a)	1	ATAN DATAN	Real Double	Real Double
End of file check on unit a	Check previous read on unit a for end-of-file. 2 is returned if none. 1 is returned if EOF.	1	EOF	Integer	Integer

Function	Definition	Number of Arguments	Symbolic Name	Type of Argument	Type of Function
Floating point fault	If a is 0, overflow is tested. If a is 1, divide fault is tested. If a is 2, underflow is tested. A 2 is returned if the condition has not occurred, a 1 otherwise.	1	IFALT	Integer	Integer
Parity error check on unit	Check previous read or write on unit a for parity error. A 2 is returned if no parity error occurred. A 1 is returned if parity error did occur.	1	IOCK	Integer	Integer

INTRINSIC FUNCTIONS

When the following functions are referenced, in-line code is generated. They may not be passed as a subprogram parameter.

Function	Definition	Number of Arguments	Symbolic Name	Type of Argument	Type of Function
Absolute value	$\lvert a \rvert$	1	ABS	Real	Real
			IABS	Integer	Integer
			DABS	Double	Double
Float	Conversion from integer to real	1	FLOAT	Integer	Real
			DFLT	Integer	Double
Fix	Conversion from real to integer	1	IFIX	Real	Integer
			DFIX	Double	Double
Transfer of sign	Sign of a_2 times $\lvert a_1 \rvert$	2	SIGN	Real	Real
			ISIGN	Integer	Integer
			DSIGN	Double	Double

Function	Definition	Number of Arguments	Symbolic Name	Type of Argument	Type of Function
Inclusive OR	Inclusive OR of I and J	2	OR(I,J)	Integer	Integer
Exclusive OR	Exclusive OR of I and J	2	EOR(I,J)	Integer	Integer
Logical product	Logical product of I and J	2	AND(I,J)	Integer	Integer
Complement	Complement (NOT) of I	1	NOT(I)	Integer	Integer
Obtain most significant part of double-precision argument		1	SNGL	Double	Real
Express single precision argument in double precision form		1	DBLE	Real	Double

STATEMENTS A

ASSEM $a_1, a_2, \ldots, a_n$

ASSIGN n_i TO $_i$

BACKSPACE i

BLOCK DATA

BYTE $(x_1, y_1(c_1=d_1), \ldots, x_n, y_n\ (c_n = d_n))$

CALL name $(c_1, c_2, \ldots, c_n)$

COMMON/name/$v_1, v_2, \ldots, v_n$

CONTINUE

DATA $v/d_1/, v_2/d_2/, \ldots, v_n/d_n/$

DIMENSION $v_1, v_2, \ldots, v_n$

DO n i = m_1, m_2, m_3

DO n i = $m_1, m_2, -m_3$

END

ENDFILE lu

EQUIVALENCE $(a_1, b_1, \ldots)$, $(a_2, b_2, \ldots), \ldots$

EXTERNAL $\text{name}_1, \text{name}_2, \ldots, \text{name}_n$

FORMAT $(\text{spec}_1, \ldots, k(\text{spec}_m, \ldots), \text{spec}_n, \ldots)$

FUNCTION name $(p_1, p_2, \ldots, p_n)$

GO TO n

GO TO i

GO TO i, $(n_1, n_2, \ldots, n_m)$

GO TO $(n_1, n_2, \ldots, n_m)$, e

IF (e) n_1, n_2, n_3

IF(l)s

OPEN

PAUSE

PAUSE n

r = e

READ (i) l

READ (i,n) l

RELATIVE $\text{name}_1, \text{name}_2, \ldots, \text{name}_n$

RETURN

REWIND i

SIGNED BYTE $(x_1 = y_1(c_1 = d_1)), \ldots, (x_n = y_n(c_n = d_n))$

SINGLE $i_1, i_2, \ldots, i_n$

STOP

STOP n

SUBROUTINE name $(p_1, p_2, \ldots, p_n)$

type $v_1, v_2, \ldots, v_n$

WRITE (i) l

WRITE (i,n) l

[illegible]

ANSI CODES FOR THE FORTRAN CHARACTER SET B

Bit Configuration	Symbol	Bit Configuration	Symbol
0100000	SP	0111011	;
0100001	!	0111100	<
0100010	"	0111101	=
0100011	#	0111110	>
0100100	$	0111111	?
0100101	%	1000000	@
0100110	&	1000001	A
0100111	'	1000010	B
0101000	(	1000011	C
0101001	)	1000100	D
0101010	*	1000101	E
0101011	+	1000110	F
0101100	,	1000111	G
0101101	-	1001000	H
0101110	.	1001001	I
0101111	/	1001010	J
0110000	0	1001011	K
0110001	1	1001100	L
0110010	2	1001101	M
0110011	3	1001110	N
0110100	4	1001111	O
0110101	5	1010000	P
0110110	6	1010001	Q
0110111	7	1010010	R
0111000	8	1010011	S
0111001	9	1010100	T
0111010	:	1010101	U

Bit Configuration	Symbol	Bit Configuration	Symbol
1010110	V	1011011	[
1010111	W	1011100	\
1011000	X	1011101	]
1011001	Y	1011110	^
1011010	Z	1011111	_

COMPILATION OUTPUT AND OPTIONS C

The 1700 Mass Storage FORTRAN compiler allows a variety of compilation options for user needs. Any combination may be used. The following defines the available options.

L	—	Source program listing with syntax checking of source code.
A	—	Object code listing with assembly language equivalences
M	—	Condensed object code listing indicating the first object code statement generated for each source statement.
R	—	Run-anywhere option allows for generation of code using relative addressing for execution in allocatable core.
K	—	ANSI FORTRAN compatibility; integers occupy two words.
X	—	Relocatable object code placed on mass memory load-and-go file for immediate execution.
P	—	Relocatable object code output to output media for retention.

All compilation processes check for syntax errors and comprehensive diagnostics are printed.

The following examples illustrate the compiler operation and listing.

L Option

Note that full compilation is not done, only a statement syntax check.

```
  1                  PROGRAM FTNOPT
             C
             C    EXAMPLE FOR FORTRAN COMPILER OPTIONS
             C
  2                  DIMENSION A(5),I(5)
  3                  DO 1 II=1,5
  4                  I(II)=II*3/A(II)
  5                1 CONTINUE
  6                  CALL SUBEXM(A,I)
  7                  J=K+6*C
  8                  IF(FUNEXM(4,9)) 10,20,10
  9               10 GO TO 20
 10               20 CONTINUE
 11                  END
```

Options LA

```
    1              PROGRAM FTNOPT
           C
           C   EXAMPLE FOR FORTRAN COMPILER OPTIONS
           C
    2              DIMENSION A(5),I(5)
    3              DO 1 II=1,5
    4              I(II)=II*3/A(II)
    5            1 CONTINUE
    6              CALL SUBEXM(A,I)
    7              J=K+6*C
    8              IF(FUNEXM(4.9)) 10,20,10
    9           10 GO TO 20
   10           20 CONTINUE
   11              END
```

```
        0000   0000                 NAM    FTNOPT
        0000   1819      FTNOPT     JMP*   .00001
        0001   000A      A          BSS       10
        000B   0005      I          BSS        5
        0010   0001      II         BSS        1
        0011   0003      0003$      CON        3
        0012   0001      J          BSS        1
        0013   0001      K          BSS        1
        0014   0006      0006$      CON        6
        0015   0002      C          BSS        2
        0017   41CE      41CE.      CON     16846
        0018   6666                 CON     26214
   3    0019   0A01      .00001     ENA        1
        001A   68F5                 STA*   II
   4    001B   0A02      .00004     ENA        2
        001C   28F3                 MUI*   II
        001D   682C                 STA*   .00005
        001E   C8F1                 LDA*   II
        001F   28F1                 MUI*   0003$
        0020   682A                 STA*   .00006
        0021   5400                 RTJ+   FLOAT
        0022   7FFF
        0023   004A P               ADC    .00006
        0024   5400                 RTJ+   FLOT
        0025   7FFF
        0026   FA40                 CON     -1471
        0027   0049 P               ADC    .00005
        0028   7FFE P               ADC    A             -2
        0029   5400                 RTJ+   Q8QFIX
        002A   7FFF
        002B   E8E4                 LDQ*   II
        002C   6ADD                 STA*   I             -I,Q
```

```
  5   002D  D8F2      1      RAC*  II
      002E  0A05             ENA        5
      002F  98E0             SUB*  II
      0030  0131             SAM        1
      0031  18E9             JMP*  .00004
  6   0032  5400             RTJ+  SUBEXM
      0033  7FFF
      0034  0001 P           ADC   A
      0035  0008 P           ADC   I
  7   0036  5CE8             RTJ*  (FLOAT )
      0037  0014 P           ADC   0006$
      0038  5CEC             RTJ*  (FLOT   )
      0039  9D40             CON   -25279
      003A  0015 P           ADC   C
      003B  004B P           ADC   .00007
      003C  5CE5             RTJ*  (FLOAT )
      003D  0013 P           ADC   K
      003E  5CE6             RTJ*  (FLOT   )
      003F  E400             CON    -7167
      0040  004B P           ADC   .00007
      0041  5CE8             RTJ*  (Q8QFIX)
      0042  68CF             STA*  J
  8   0043  5400             RTJ+  FUNEXM
      0044  7FFF
      0045  0017 P           ADC   41CE.
      0046  C0C5             CON   -16186
      0047  0105             SAZ        5
  9   0048  1805      10     JMP*  20
      0049  0001      .00005 BSS        1
      004A  0001      .00006 BSS        1
      004B  0002      .00007 BSS        2
 11   004D  5400      20     RTJ+  Q8STP
      004E  7FFF
 11   0000  0000             END        0

PROGRAM LENGTH $004F (     79)

OPTS = AL

EXTERNALS
Q8QFIX FLOT   Q8STP  FLOAT  SUBEXM FUNEXM
```

Options LM

Note condensed object code listing. This form is useful when the list device is a Teletype.

```
      1            PROGRAM FTNOPT
            C
            C    EXAMPLE FOR FORTRAN COMPILER OPTIONS
            C
      2            DIMENSION A(5),I(5)
      3            DO 1 II=1,5
      4            I(II)=II*3/A(II)
      5          1 CONTINUE
      6            CALL SUBEXM(A,I)
      7            J=K*6*C
      8            IF(FUNEXM(4,9)) 10,20,10
      9         10 GO TO 20
     10         20 CONTINUE
     11            END



      3    0019  0A01      .00001  ENA          1
      4    001B  0A02      .00004  ENA          2
      5    002D  D8E2      1       RAO*   II
      6    0032  5400              RTJ+   SUBEXM
      7    0036  5CFB              RTJ*   (FLOAT )
      8    0043  5400              RTJ+   FUNEXM
      9    004B  1805      10      JMP*   20
     11    004D  5400      20      RTJ+   Q8STP
     11    0000  0000              END          0

  PROGRAM LENGTH $004F (     79)

  OPTS = LM

  EXTERNALS
  Q8QFIX FLOT   Q8STP  FLOAT  SUBEXM FUNEXM
```

Options LAR

Note that no program relocatable addresses are generated; hence, the program is able to run in allocatable core.

```
     1              PROGRAM FTNØPT
             C
             C    EXAMPLE FOR FØRTRAN COMPILER OPTIONS
             C
     2              DIMENSION A(5),I(5)
     3              DO 1 II=1,5
     4              I(II)=II*3/A(II)
     5            1 CONTINUE
     6              CALL SUBEXM(A,I)
     7              J=K*6*C
     8              IF(FUNEXM(4,9)) 10,20,10
     9           10 GO TO 20
    10           20 CONTINUE
    11              END
```

```
          0000  0000                NAM    FTNOPT
                          .00001
          0000  1819      FTNOPT    JMP*   .00002
          0001  000A      A         BSS        10
          000B  0005      I         BSS         5
          0010  0001      II        BSS         1
          0011  0003      0003$     CON         3
          0012  0001      J         BSS         1
          0013  0001      K         BSS         1
          0014  0006      0006$     CON         6
          0015  0002      C         BSS         2
          0017  41CE      41CE.     CON     16846
          0018  6666                CON     26214
          0019  5802      .00002    RTJ*   .00005
          001A  FFE5                ADC    .00001
          001B  0001      .00005    BSS         1
          001C  C8FE                LDA*   .00005
          001D  88FC                ADD*   .00005     -Ī
          001E  68FC                STA*   .00005
     3    001F  0A01                ENA         1
          0020  68EF                STA*   II
```

```
 4   0021  0A02    .00006  ENA        2
     0022  28ED            MUI*   II
     0023  682C            STA*   .00007
     0024  C8EB            LDA*   II
     0025  28EB            MUI*   0003$
     0026  682A            STA*   .00008
     0027  5400            RTJ+   FLOAT
     0028  7FFF
     0029  8027            ADC    .00008
     002A  5400            RTJ+   FLOT
     002B  7FFF
     002C  5FA4            CON     24484
     002D  0022            ADC    .00007
     002E  7FD0            ADC    A          -2
     002F  5400            RTJ+   Q8QFIX
     0030  7FFF
     0031  E8DE            LDQ*   II
     0032  6AD7            STA*   I          -I,Q
 5   0033  D8DC    1       RAO*   II
     0034  0A05            ENA        5
     0035  98DA            SUB*   II
     0036  0131            SAM        1
     0037  18E9            JMP*   .00006
 6   0038  5400            RTJ+   SUBEXM
     0039  7FFF
     003A  FFC6            ADC    A
     003B  FFCF            ADC    I
 7   003C  5CEB            RTJ*   (FLOAT )
     003D  FFD6            ADC    0006$
     003E  5CEC            RTJ*   (FLOT  )
     003F  59D4            CON     22996
     0040  7FD4            ADC    C
     0041  0010            ADC    .00009
     0042  5CE5            RTJ*   (FLOAT )
     0043  FFCF            ADC    K
     0044  5CE6            RTJ*   (FLOT  )
     0045  5E40            CON     24128
     0046  000B            ADC    .00009
     0047  5CEB            RTJ*   (Q8QFIX)
     0048  68C9            STA*   J
 8   0049  5400            RTJ+   FUNEXM
     004A  7FFF
     004B  FFCB            ADC    41CE.
     004C  C0C5            CON    -16186
     004D  0105            SAZ        5
 9   004E  1805    10      JMP*   20
     004F  0001    .00007  BSS        1
     0050  0001    .00008  BSS        1
     0051  0002    .00009  BSS        2
11   0053  5400    20      RTJ+   Q8STP
     0054  7FFF
11   0000  0000            END        0

PROGRAM LENGTH $0055 (     85)

OPTS = RAL

EXTERNALS
Q8QFIX FLOT   Q8STP  FLOAT  SUBEXM FUNEXM
```

Options LAK

This form allocates two words of memory for each integer. The actual executable code only uses one of the two words.

```
     1            PROGRAM FTNOPT
           C
           C    EXAMPLE FOR FORTRAN COMPILER OPTIONS
           C
     2            DIMENSION A(5),I(5)
     3            DO 1 II=1,5
     4            I(II)=II*3/A(II)
     5          1 CONTINUE
     6            CALL SUBEXM(A,I)
     7            J=K*6*C
     8            IF(FUNEXM(4,9)) 10,20,10
     9         10 GO TO 20
    10         20 CONTINUE
    11            END


        0000  0000               NAM   FTNOPT
        0000  1821      FTNOPT   JMP*  .00001
        0001  000A      A        BSS       10
        000B  000A      I        BSS       10
        0015  0002      II       BSS        2
        0017  0003      0003$    CON        3
        0018  0002      J        BSS        2
        001A  0002      K        BSS        2
        001C  0006      0006$    CON        6
        001D  0002      C        BSS        2
        001F  41CE      41CE.    CON    16846
        0020  6666               CON    26214
     3  0021  0A01      .00001   ENA        1
        0022  68F2               STA*  II
     4  0023  0A02      .00004   ENA        2
        0024  28F0               MUI*  II
        0025  682F               STA*  .00005
        0026  0A02               ENA        2
        0027  28ED               MUI*  II
        0028  682D               STA*  .00006
        0029  C8EB               LDA*  II
        002A  28EC               MUI*  0003$
        002B  682B               STA*  .00007
        002C  5400               RTJ*  FLOAT
        002D  7FFF
        002E  0056 P             ADC   .00007
        002F  5400               RTJ*  FLOT
        0030  7FFF
        0031  FA40               CON   -1471
        0032  0055 P             ADC   .00006
        0033  7FFE P             ADC   A            -2
        0034  5400               RTJ*  Q8QFIX
        0035  7FFF
        0036  F81F               LDQ*  .00005
        0037  6AD1               STA*  I            -2,Q
```

```
  5    0038  D8DC      1          RAO*   II
       0039  0A05                 ENA        5
       003A  98DA                 SUB*   II
       003B  0131                 SAM        1
       003C  18E6                 JMP*   .00004
  6    003D  5400                 RTJ+   SUBEXM
       003E  7FFF
       003F  0001 P               ADC    A
       0040  000B P               ADC    I
  7    0041  5CEB                 RTJ*   (FLOAT )
       0042  001C P               ADC    0006$
       0043  5CEC                 RTJ*   (FLOT   )
       0044  9D40                 CON    -25279
       0045  001D P               ADC    C
       0046  0057 P               ADC    .00008
       0047  5CE5                 RTJ*   (FLOAT )
       0048  001A P               ADC    K
       0049  5CE6                 RTJ*   (FLOT   )
       004A  F400                 CON     -7167
       004B  0057 P               ADC    .00008
       004C  5CE8                 RTJ*   (Q8QFIX)
       004D  68CA                 STA*   J
  8    004E  5400                 RTJ+   FUNEXM
       004F  7FFF
       0050  001F P               ADC    41CE.
       0051  C0C5                 CON    -16186
       0052  0106                 SAZ        6
  9    0053  1806      10         JMP*   20
       0054  0001      .00005     BSS        1
       0055  0001      .00006     BSS        1
       0056  0001      .00007     BSS        1
       0057  0002      .00008     BSS        2
 11    0059  5400      20         RTJ+   Q8STP
       005A  7FFF
 11    0000  0000                 END        0

PROGRAM LENGTH $005B (     91)

OPTS = KAL

EXTERNALS
Q8QFIX FLOT   Q8STP  FLOAT  SUBEXM FUNEXM
```

Options LX

Note that the full compilation has taken place.

```
      1              PROGRAM FTNOPT
                C
                C   EXAMPLE FOR FORTRAN COMPILER OPTIONS
                C
      2              DIMENSION A(5),I(5)
      3              DO 1 II=1,5
      4              I(II)=II*3/A(II)
      5            1 CONTINUE
      6              CALL SUBEXM(A,I)
      7              J=K+6*C
      8              IF(FUNEXM(4.9)) 10,20,10
      9           10 GO TO 20
     10           20 CONTINUE
     11              END

PROGRAM LENGTH $004F (     79)

OPTS = LX

EXTERNALS
Q8QFIX FLOT   Q8STP  FLOAT  SUBEXM FUNEXM
```

Options PX

Note that no listing output is generated, but full compilation has occurred with object and load and go output.

```
    OPTS = PX
```

REFERENCE MANUALS D

	Title	Publication Number
1700	MSOS 4 Reference Manual	60361500
1700	MSOS 4 Macro Assembler Reference Manual	60361900
1700	MSOS 4 Mass Storage FORTRAN Version 3 Reference Manual	60362000
1700	MSOS 4 Computer System Codes	60163500
1700	MSOS 4 Macro Assembler General Information Manual	39519800
1700	MSOS 4 Small Computer Maintenance Monitor Reference Manual	39520200
1700	MSOS 4 Instant	39520500
1700	MSOS 4 File Manager Version 1 Reference Manual	39520600
1700	MSOS 4 Installation Handbook	39520900
1700	MSOS 4 Small Computer Maintenance Monitor Instant	39521700
1700	MSOS 4 General Information Manual	39522400

COMMENT SHEET

MANUAL TITLE CONTROL DATA® 1700 Computer Systems 1700 MSOS 4

Mass Storage FORTRAN Version 3A/B General Information Manual

PUBLICATION NO. 39519900 REVISION A

FROM NAME: ______________________

BUSINESS ADDRESS: ______________________

COMMENTS: This form is not intended to be used as an order blank. Your evaluation of this manual will be welcomed by Control Data Corporation. Any errors, suggested additions or deletions, or general comments may be made below. Please include page number.

CUT ALONG LINE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
PUBLICATIONS AND GRAPHICS DIVISION
4455 EASTGATE MALL
LA JOLLA, CALIFORNIA 92037

CUT ALONG LINE

FOLD FOLD