CONTROL DATA
CORPORATION

CONTROL DATA®
6000 COMPUTER SYSTEMS
7600 COMPUTER SYSTEM

FORTRAN EXTENDED INSTANT
6000 VERSION 3
7600 VERSION 1

CONTROL DATA
CORPORATION

CONTROL DATA®
6000 COMPUTER SYSTEMS
7600 COMPUTER SYSTEM

FORTRAN EXTENDED INSTANT
6000 VERSION 3
7600 VERSION 1

REVISION RECORD	
REVISION	**DESCRIPTION**
A (5-28-71)	Original printing.
Publication No. 60305900	

Additional copies of this manual may be obtained from the nearest Control Data Corporation sales office.

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Documentation Department
215 MOFFETT PARK DRIVE
SUNNYVALE, CALIFORNIA 94086

INTRODUCTION

FORTRAN Extended 6000 version 3/7000 version 1 processes a scientific language for programming on Control Data Corporation® 6000 and 7000 Series Computers under control of the SCOPE Operating System.

This Instant provides a short reference tool for frequently used information. Detailed information on FORTRAN Extended and the other languages mentioned is available in the applicable reference manuals.

FORTRAN STATEMENTS & CODING FORM

FORTRAN Extended source programs consist of an ordered set of statements from which the compiler generates machine instructions and constants. Each line of a FORTRAN coding form corresponds to a punched card containing one statement line. Statements are written in the following columns:

	Column	Content
Statements	1-5	Statement label
	6	Blank or zero
	7-72	FORTRAN statement
	73-80	Identification field
Continuations	1-5	Ignored
	6	Non-zero, non-blank FORTRAN character
	7-72	Continued statement
	73-80	Identification field
Comments	1	C or $ or *
	2-80	Comments

CONTROL CARD FORMAT

FTN.comments

FTN (p_1,p_2,...,p_n)

FTN. is equivalent to:

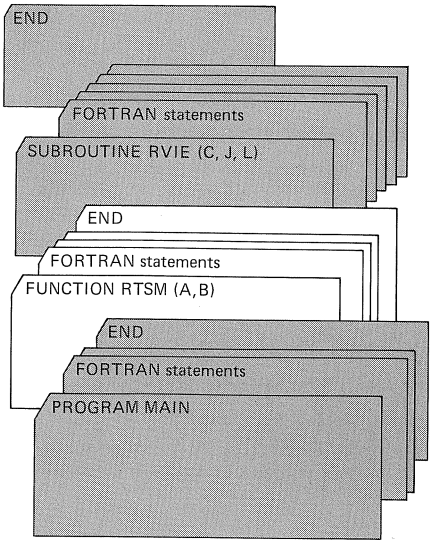
FTN (I=INPUT,L=OUTPUT,B=LGO,S=SYSTEXT,OPT=1,R=1)

The parameter list may include the following options:

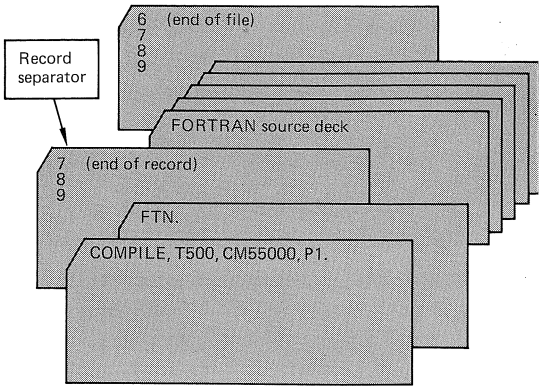
Control Card Option	Usage	Release Setting
A	Abort to EXIT(S) card if fatal errors encountered during compilation	No abort
B	Produce object code file	Object code on standard file (LGO)
C	Use COMPASS assembler for compiler generated code (almost trebles amount of central processor time for compilation)	FORTRAN assembler is used
D	Debug mode of compilation	No debug mode
E	Format file for editing (COMPASS card images file is produced with *DECK cards for each program unit, suitable as input for UPDATE)	No file for editing
G	Compile and go option	No compile and go
I=lfn	Select compiler input file	lfn=INPUT
list=lfn	Select compiler listing file (lfn) and listing option (list):	lfn=OUTPUT list=L
	L List source code	L
	O List COMPASS card images	No
	X List ANSI extension diagnostics	No
	N Suppress informative diagnostics	No
	R (equivalent to long reference map option R=2)	No

Control Card Option	Usage	Release Setting
Reference Map Level	Kind of reference map:	R = 1 unless L option = 0; then R = 0
R=0	No map	
=1	Short map (symbols, addresses, properties)	
=2	Long map (short map, references by line numbers, DO-loop map)	
=3	Long map, printed common block members and equivalence classes	
OPT=level	Select level of optimization: 0 lowest optimization 1 slightly above FORTRAN Extended 2.0 optimization 2 program unit flow analysis used in optimization	OPT = 1
ROUND=s	s = * / + - 1-4 operators to indicate rounded arithmetic	No rounding
SYSEDIT=ss	For system programmer usage:	No search
	ss=FILES, form execution time input/output unit references through indirect search of low core table rather than by using entry points and external references	
	ss=IDENT, append a $ to IDENT name of recognized routines in addition to ENTRY point name. Implies a special selection for users of SEGMENT feature since segments are specified by IDENT names	No special selection
	SYSEDIT only is equivalent to SYSEDIT = FILES and IDENT	
T	Maximum error checking in mathematical library routines (basic external functions)	Maximum checking not carried out
V	Selects minimal I/O buffer allocation (513 word) for compilation. May increase compile time, but jobs with large numbers of declarative statements compile in smaller field length than otherwise	

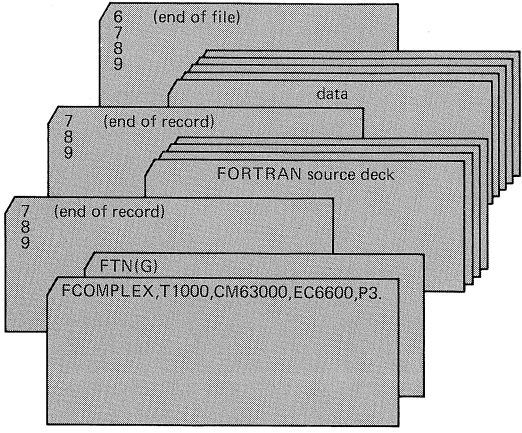
SOURCE DECK


END
FORTRAN statements
SUBROUTINE RVIE (C, J, L)
END
FORTRAN statements
FUNCTION RTSM (A, B)
END
FORTRAN statements
PROGRAM MAIN

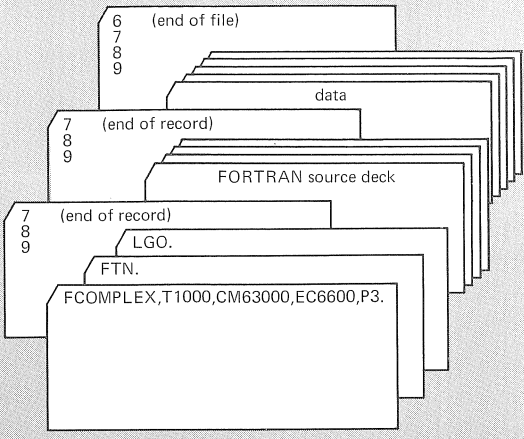
COMPILATION



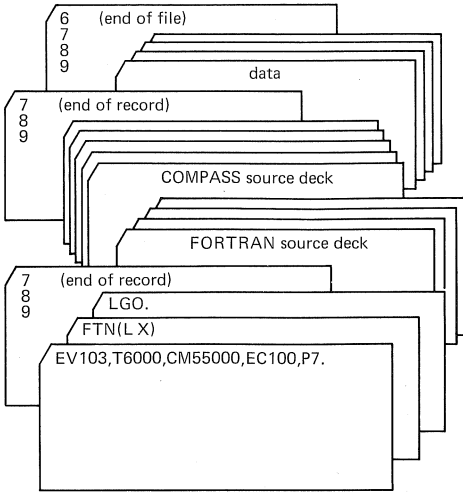
COMPILATION & EXECUTION



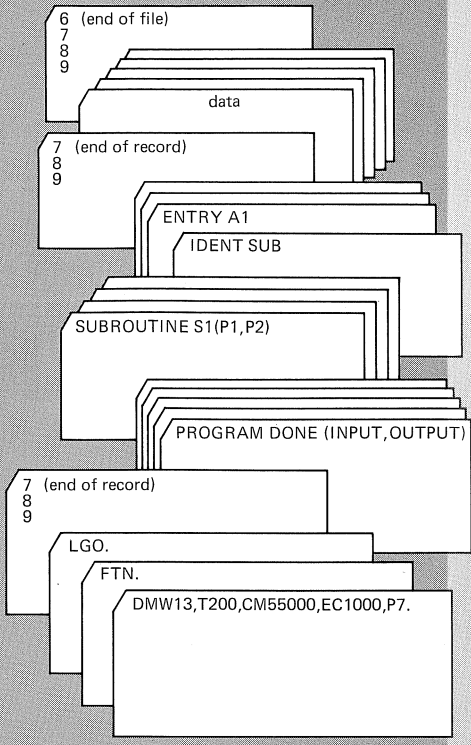
COMPILATION & EXECUTION



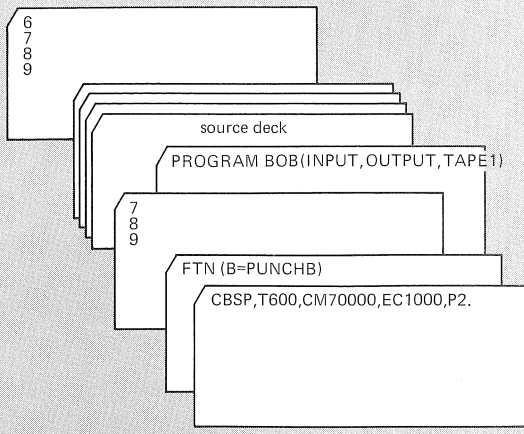
FORTRAN COMPILATION, COMPASS ASSEMBLY & EXECUTION

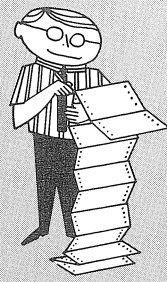


COMPILE & EXECUTE WITH SUBROUTINE & COMPASS

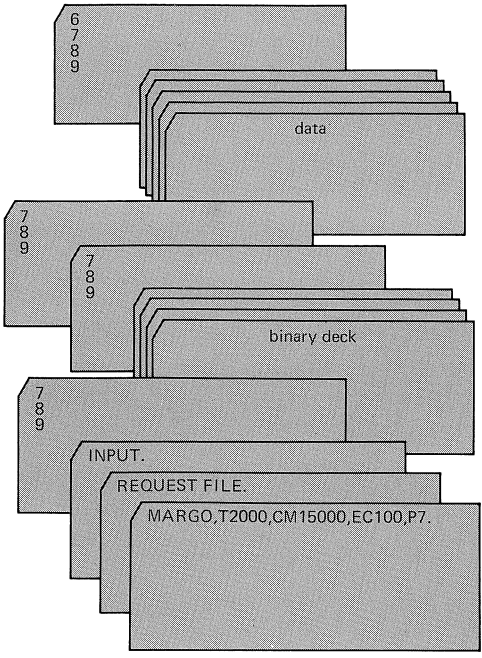


COMPILE & PRODUCE BINARY CARDS

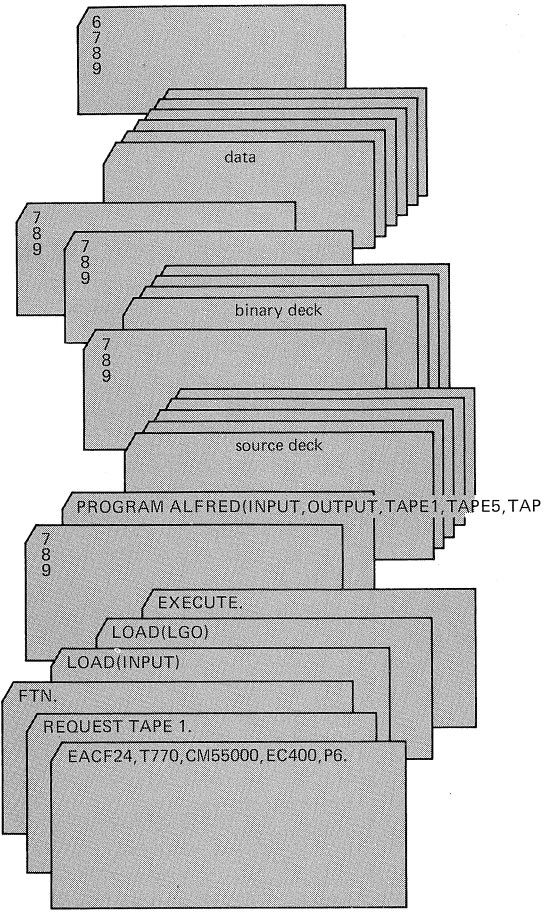




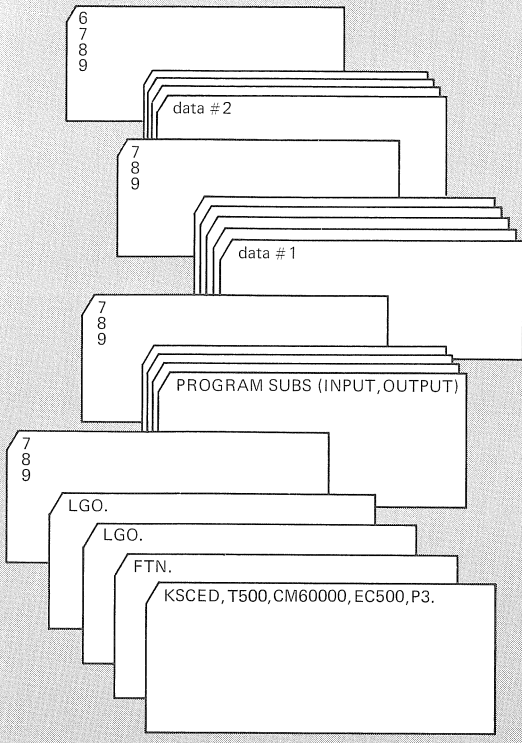
LOAD & EXECUTE BINARY PROGRAM



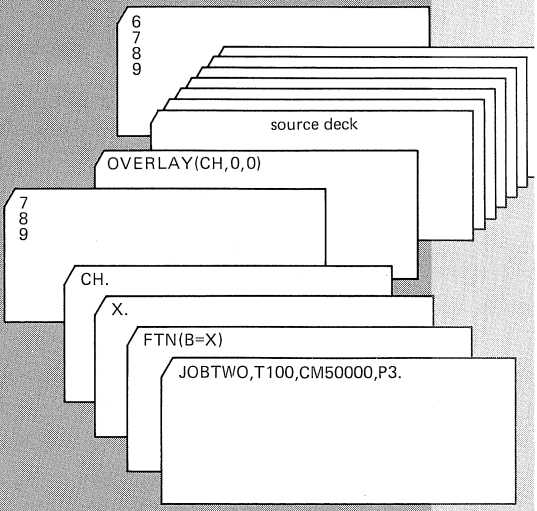
COMPILE & EXECUTE WITH RELOCATABLE BINARY DECK



COMPILE ONCE & EXECUTE WITH DIFFERENT DATA DECKS



COMPILATION & 2 EXECUTIONS WITH OVERLAYS



EXPRESSIONS

Arithmetic, masking, relational, and logical operations may be specified for the evaluation of expressions.

Arithmetic	Expression using operands of types other than logical
Masking	Arithmetic expression using one of the operators: .OR. .AND. .NOT.
Relational	Arithmetic or masking expressions separated by relational operators
Logical	Logical variables, functions, constants, and relational expressions separated by logical operators

NAME TYPE

Type of variable names, array names, and statement function names may be indicated implicitly or explicitly.

Implicit typing:	Determined by initial letter of name. Real: A-H, O-Z. Integer: I, J, K, L, M, N
Explicit typing:	Determined by declaration with or without the prefix TYPE:

REAL DOUBLE or DOUBLE PRECISION

INTEGER LOGICAL

COMPLEX ECS

CONSTANTS

Data Type	Characteristics
Integer	1-18 digits, $0-(2^{59}-1)$, no decimal point
Real	1-15 digits, with decimal point, optional exponent base 10
Double Precision	1-29 digits, decimal point, optional exponent base 10 which may be signed

Data Type	Characteristics
Complex	Real part and imaginary part, decimal point, optionally signed, real numbers
Octal	1-20 octal digits with B suffix, no sign or decimal point
Logical	.TRUE. .T. .FALSE. .F. true is -1, false is all zero bits
Hollerith	nHf, nRf (right justified), nLf (left justified),

n number of characters in string

H left justified, blank fill

R right justified, zero fill

L left justified, zero fill

f characters in string (embedded blanks included); 1-10 in replacement statement, 1-150 in FORMAT statement; DATA statement and parameter list of a call may extend through 19 continuation lines

VARIABLES & ARRAYS

Data Type	Variables	Arrays
Integer Real	Implicit or explicit type single variable name	Implicit or explicit type array name with subscripts in parentheses
Double Precision Complex Logical	Explicit type single variable name	Explicit type array name with subscripts in parentheses

NAME: An alphabetic character followed by 0-6 alphanumeric characters.

SUBSCRIPTS AND ARRAYS

A subscript is written as a list of subscript expressions. An array element is an alphanumeric identifier that is the name of an array, followed by up to three subscript expressions representing a single element within the array. A subscript expression may be any legal arithmetic expression.

Subscript expressions are separated by a comma and the subscript is enclosed in parentheses. If the number of subscript expressions in a reference is less than the declared dimensionality, the compiler assumes missing subscripts have a value of one.

A(2)≡A(2,1)

A(,2)≠A(1,2) illegal form

A subscript expression may be missing from the right only; if a comma appears, it must be preceded and followed by a subscript expression.

If the subscript list does not appear, all subscript expressions are assumed to be one, and an informative diagnostic is issued. If the subscript expression is not integer, the value will be truncated to integer.

Standard form: (constant * variable ± constant)

For DIMENSION A(L,M,N) the location of A(i,j,k) with respect to A(1,1,1) is determined by the formula:

A+(i-1+L*(j-1+M*(k-1)))*e

where e is the number of words occupied by each element of A. For double precision and complex, e=2; otherwise, e=1.

DATA DECLARATION AND STORAGE ALLOCATION

Key:

- v Variable (all can be v_i (i_i) also)
- i Integer variables or integer constants (maximum 3)
- x Common block name (may be null)
- a List of variable names
- k Data variable list (optionally DO-implied)
- d Data constant list
- r Data variable list

INTEGER $v_1,v_2,\ldots,v_n$

REAL $v_1,v_2,\ldots,v_n$

DOUBLE PRECISION $v_1,v_2,\ldots,v_n$

COMPLEX $v_1,v_2,\ldots,v_n$

LOGICAL $v_1,v_2,\ldots,v_n$

†ECS $v_1,v_2,\ldots,v_n$

†TYPE INTEGER $v_1,v_2,\ldots,v_n$

†TYPE REAL $v_1,v_2,\ldots,v_n$

†TYPE DOUBLE PRECISION $v_1,v_2,\ldots,v_n$

†TYPE COMPLEX $v_1,v_2,\ldots,v_n$

†TYPE LOGICAL $v_1,v_2,\ldots,v_n$

†TYPE ECS $v_1,v_2,\ldots,v_n$

DIMENSION $v_1(i_1),v_2(i_2),\ldots,v_n(i_n)$

COMMON $/x_1/a_1/\ldots/x_n/a_n$

EQUIVALENCE $(k_1),(k_2),\ldots,(k_n)$

DATA $k_1/d_1/,k_2/d_2/,\ldots,k_n/d_n/$

†DATA $(r_1=d_1),(r_2=d_2),\ldots,(r_n=d_n)$

---

† permitted non-ANSI form

REPLACEMENT STATEMENT

v = e

- v Variable or array element
- e Arithmetic, logical, or masking expression

CONTROL STATEMENTS

Control statements alter sequential execution of statements, perform tests and iterations, and terminate subprograms and programs.

Form	Example
DO	
DO n i = m_1,m_2,m_3	DO 50 K = 3,18,6
DO n i = m_1,m_2	DO 50 K = 3,18
n Terminal statement label	
i Integer control variable	
m_1 Initial value	
m_2 Terminal value	
m_3 Incremental (maximum value of running index is $2^{17}-2$)	
If m_3 is omitted, a value of one is assumed.	
IF	
IF (e)k_1,k_2,k_3	IF (J–M**2)5,15,2
e Variable or arithmetic expression of integer, real, double precision, or complex type	
k Statement labels	
e < 0 control transfers to k_1	
e = 0 control transfers to k_2	
e > 0 control transfers to k_3	
IF (e)k_1,k_2	IF (X/(D**4))6,14
e Masking or arithmetic expression	
e ≠ 0 control transfers to k_1	
e = 0 control transfers to k_2	

ANSI equivalent is:

IF(e)k_1,k_2,k_1

Form	Example
IF (e)s e Logical expression s Any executable statement except DO or logical IF. If e is false, s is not executed; if e is true, s is executed.	IF (X.LE.B) GO TO 73

IF (e)k_1,k_2
e Logical expression
k Statement labels
If e is true, transfer to k_1; if e is false, transfer to k_2
ANSI equivalent is:
IF (ZR .GT.F) GOTO31
GO TO 35
	IF (ZR .GT. F)31,35

GO TO

GO TO k k Statement label	GO TO 500
ASSIGN k TO i k Executable statement label i Integer variable	ASSIGN 25 TO I
then GO TO i,(k_1,k_2,...,k_n) k Statement labels i Integer variable; must contain value assigned by preceding ASSIGN statement and be a label in list. When i $\leq$ 0 or i > maximum, a fatal error occurs at execution.	ASSIGN 7 to JNP GO TO JNP,(4,2,7) 7 DKW(K) = DKW(K+5)
GO TO (k_1,k_2,...,k_n), i k Statement labels i Variable or expression	GO TO (3,7,4,9),JL

Form	Example
CONTINUE	
CONTINUE (usually preceded by statement label)	100 CONTINUE
PAUSE	
PAUSE	PAUSE
PAUSE n	PAUSE 23456
n String of 1–5 octal digits	
STOP	
STOP	STOP
STOP n	STOP 55675
n String of 1–5 octal digits without a B suffix	
CALL	
CALL s (a_1,a_2,...,a_n)	CALL X (A,B,C,D)
CALL s	CALL X
CALL s (a_1...a_n),RETURNS(b_1...b_n)	CALL X (A,B),RETURNS(35,62)
CALL s, RETURNS (b_1,b_2,...,b_n)	CALL X, RETURNS (5,9)
s Subroutine name	
a Actual parameters	
b Statement labels in calling program or subprogram	
RETURN	
RETURN	RETURN
RETURN a	RETURN CHI
a Parameter in RETURNS list	
END	
END (only characters on card; anywhere in columns 7–72)	END

INPUT/OUTPUT

Input/output statements control the flow of data. Two modes are possible: formatted (coded) and unformatted (binary).

Parameters

u Identifies I/O unit; integer constant or simple integer variable

f Identifies format specification; FORMAT label or array name

k I/O list of data to be transferred

Formatted Input/Output

Form	Example
READ (u,f)k	READ (10,15)A,B,CF,R
READ (u,f)	READ (1,4)
†READ f,k	READ 10,LISTX,BLOK,WEB
†READ f	READ 25
WRITE (u,f)k	WRITE (10,15)B,(A(I),I,I=1,100),C,X,Y,Z
WRITE (u,f)	WRITE (14,50)
†PRINT f,k	PRINT 12,ADS,KLM
†PRINT f	PRINT 25
†PUNCH f,k	PUNCH 30,HITE
†PUNCH f	PUNCH 50

†Non-ANSI forms

NAMELIST

Form	Example
NAMELIST /y_1/a_1.../y_n/a_n	NAMELIST /X1/R1,R3/X2/R2
y NAMELIST name	NAMELIST /SL01/D1,D2,L5
a List of variables or array names	

Forms of input data for use with namelist

$NAME V = C	J = 52
a = $d_1,\ldots,d_j$	B = 12,5.6,7*32
a(n) = $d_1,\ldots,d_m$	XAP (6,9) = 73.2E4,45,9*21
NAME Corresponds to y (above)	

Form	Example
Forms of input data for use with namelist (Continued)	
v Variable name	
c Constant	
a Array	
n Integer constant subscript	
d Simple constants or repeated constants of form k*c where k is repetition factor	
Namelist I/O	
Read (u,y)	READ (5,XI)
Write (u,y)	WRITE (10,CONTS)
Read y	READ SECD
Print y	PRINT ADDED
Punch y	PUNCH TRY 14

Unformatted Input/Output

Form	Example
READ (u)k	READ (35)DATA,INFO
READ (u)	READ (10)
WRITE (u)k	WRITE (15) CAN,LOCA,PLP
WRITE (u)	WRITE (5)
REWIND u	REWIND M or REWIND 3
BACKSPACE u	BACKSPACE I or BACKSPACE 10
ENDFILE u	ENDFILE J or ENDFILE 9
ECS I/O	
CALL READEC (a,b,n)	CALL READEC (D,J(4,1,3),17)
CALL WRITEC (a,b,n)	CALL WRITEC (XY,G,50)
a Variable or array element in central memory	
b Variable or array element in ECS common block	
n Integer constant or integer expression	

Form	Example
Mass Storage I/O	
CALL OPENMS (u,ix,l,p)	CALL OPENMS (1,INDEX,100,0)
CALL READMS (u,fwa,n,i)	CALL READMS (1,RECORD,N,IND)
CALL WRITMS (u,fwa,n,i)	CALL WRITMS (1,RECORD,N,IND)
CALL STINDX (u,ix,l)	CALL STINDX (1,WIND,50)

u Logical unit number

ix First word address of index (in CM)

l Length of index
- l ≥ 2 (number of index entries) +1 for name index
- l ≥ number of entries +1 for number index

fwa First word address

n Number of CM words to be transferred

i Record number/cell address containing record name or number

---

Buffered I/O

BUFFER IN (u,p)(fwa,lwa)	BUFFER IN (1,1)(A(1),A(LEN))
BUFFER OUT (u,p)(fwa,lwa)	BUFFER OUT (I,J)(B(L1),B(L2))

u Logical unit number

p Recording mode

fwa First word address

lwa Last word address

Encode/Decode Statements

ENCODE (n,f,A)k	ENCODE (30,2,BETA)X,Y
DECODE (n,f,A)k	DECODE (15,1,DELTA)R5,C7

n Number of characters in record ((n+9)/10 words long)

f FORMAT statement

A Identifier, variable, or array which supplies starting location

k I/O list

DEBUGGING FACILITY

The debugging mode of compilation, along with the source cross-reference map selection, is provided specifically to aid in the development or conversion of programs. In the debugging mode of compilation, a programmer can establish a record of selected operations as they are performed in the execution of his program. This mode facilitates debugging from a source listing, and perhaps a source cross-reference map; if core dumps are required, interpretation is much easier.

Features provided with the debugging mode of compilation:

Array bounds checking

Program flow tracing

Call and return tracing

Function call and value returned tracing

Stores checking

Assigned GO TO checking

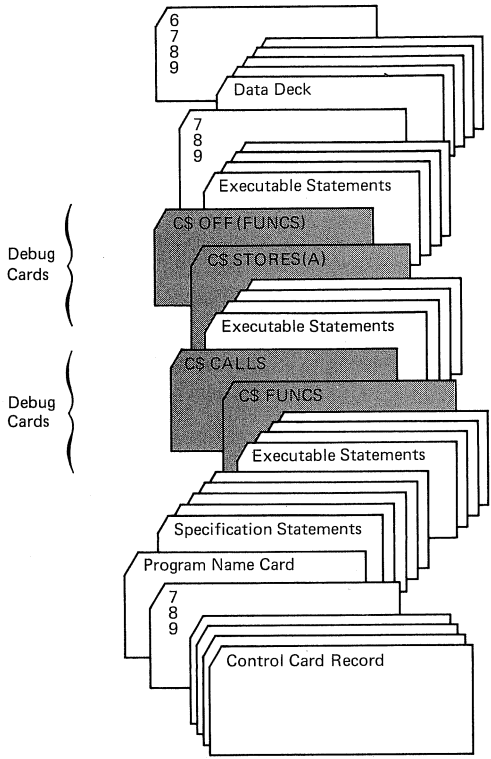
Partial execution of routines containing fatal errors

The debugging mode is selected by the option D on the FTN control card. In this mode, debugging selection cards are recognized. If this mode is not specified, debugging selection cards are treated as comments.

In the debugging mode, a program is compiled so that specified checks can be performed during execution; however, execution will stop when a fatal error is detected.

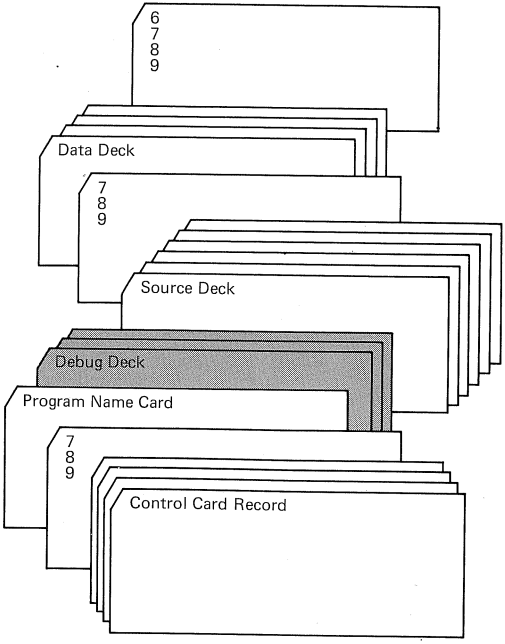
When a program is compiled in debug mode, 12000_8 words will be required beyond the minimum field length for non-debug mode compilation. To execute, 2500_8 words beyond the minimum will be required.

Individual Debug Cards Interspersed in Program Unit



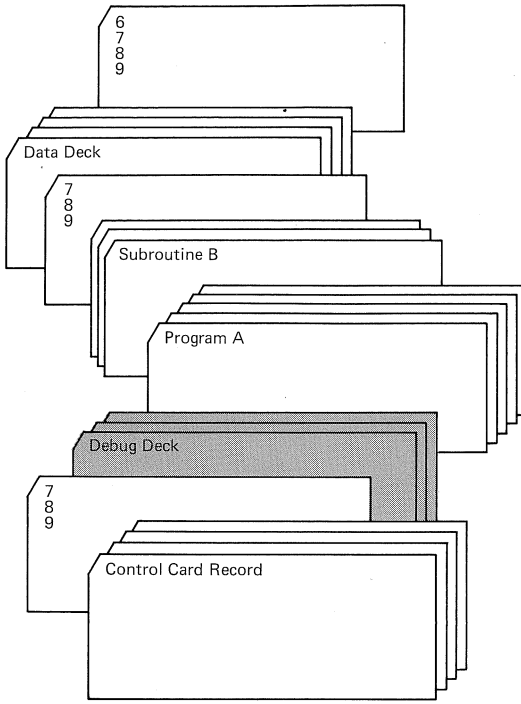
This positioning is especially useful when a new program is run for the first time and the accuracy of specific areas, such as array bounds, is in doubt.

Placed Immediately After Program Name Card and Before Specification Statements



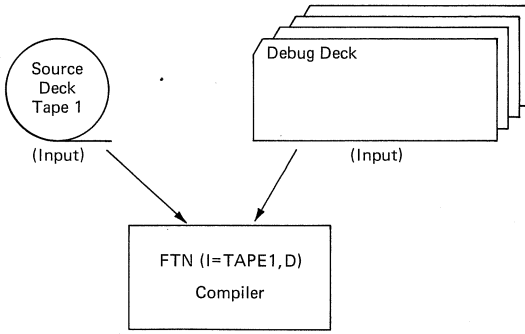
All statements in program unit will be debugged (unless limiting bounds are specified in the debug deck), but no statements in other program units will be debugged. Especially useful when several program units are known to be free of bugs but one unit is new or known to have bugs.

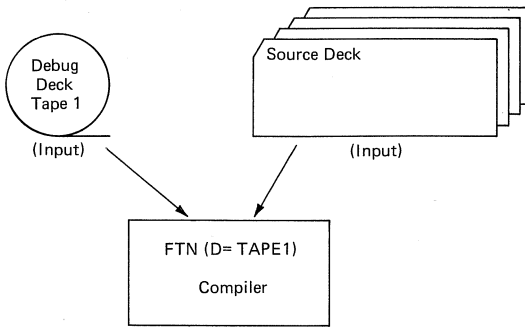
Placed Immediately in Front of First Source Line
(When D file is same as source input file)



All program units (Program A and Subroutine B) will be debugged (unless exclusive program units are specified in the debug deck). Particularly useful when a program is run for the first time, since it ensures that all program units will be debugged.

Debug Deck on Separate File (External Debug Deck)





All program units are debugged (unless exclusive program units are specified in the deck). Several jobs can be debugged using the same debugging deck.

Debugging statements follow all normal FTN formation rules, with the additional requirement that columns 1 and 2 of each statement line, including continued statements, must contain C$. Columns 3-5 of continuation lines must be blank. If no parameters are specified, all applicable statements are checked except STORES.

Name	Function	Format
DEBUG	Initiates debugging; must begin with this statement	C$ DEBUG($name_1, \ldots, name_n$) C$ DEBUG
AREA	Designates specific area to be checked in a program unit.	C$ DEBUG C$ AREA($bounds_1$),...,($bounds_n$) or C$ DEBUG($name_1, \ldots, name_n$) C$ AREA/$name_1$/(bounds), .../$name_n$/(bounds),...
CALLS	Traces calls to and returns from specified subroutines.	C$ CALLS($a_1, \ldots, a_n$) C$ CALLS
FUNCS	Traces usage of functions in a program unit.	C$ FUNCS($a_1, \ldots, a_n$) C$ FUNCS
GO TOS	Assures that the selected statement numbers are valid when executed in an assigned GO TO.	C$ GOTOS
NOGO	Suppresses partial execution of a compiled routine whenever a fatal compilation error occurs.	C$ NOGO
STORES	Records stores into specified variables as a result of replacement statements;	C$ STORES($c_1, c_2, \ldots, c_n$)

Name	Function	Format
STORES (Cont)	c Can be variable names or relational expressions in either of the forms: variable .relational operator. constant variable { .valid. (out of range or indefinite) .range. (out of range) .indef. (indefinite)	
TRACE	Produces a message for each intraprogram transfer in control at a DO-nest level $\leq$ level specified.	C$ TRACE(level) C$ TRACE
OFF	Suppresses interspersed debugging statements at compilation time.	C$ OFF($x_1, \ldots, x_n$) C$ OFF

CROSS REFERENCE MAP

The cross reference map produced of all programmer created symbols in a program unit is determined by the R option on the control card.

R = 0	No map
= 1	Short map (symbols, addresses, properties)
= 2	Long map (short map, references by line number, and DO-loop map)
= 3	Long map and printout of common block members and equivalence classes
Unspecified	Implies R = 1

The default option is R = 1 unless the L option equals 0; then R = 0.

Intrinsic Function		Argument/Function
ABS(x)	Absolute value	Real/real
AIMAG(c)	Imaginary part of complex argument	Complex/real
AINT(x)	Truncation	Real/real
AMAX0(i_1, i_2, ...)	Maximum argument	Integer/real
AMAX1(x_1, x_2, ...)	Maximum argument	Real/real
AMIN0(i_1, i_2, ...)	Minimum argument	Integer/real
AMIN(x_1, x_2, ...)	Minimum argument	Real/real
AMOD(x_1, x_2)	x_1 modulo x_2	Real/real
AND(x_1, x_2)	Logical product	Single word
CMPLX(x_1, x_2)	$x_1 + ix_2$	Real/complex
COMPL(x)	Complement of x	Single word
CONJG(c)	Conjugate of c	Complex/complex
DABS(x)	Absolute value	Double/double
DIM(x_1, x_2)	$x_1 - \text{Min}(x_1, x_2)$	Real/real
DBLE(x)	Conversion	Real/double
DMAX1(d_1, d_2, ...)	Maximum argument	Double/double
DMIN1(d_1, d_2, ...)	Minimum argument	Double/double
DMOD(d_1, d_2)	d_1 modulo d_2	Double/double
DSIGN(d_1, d_2)	Sign d_2 times d_1	Double/double
FLOAT(i)	Conversion	Integer/real

Intrinsic Function		Argument/Function
IABS(i)	Absolute value	Integer/integer
IDIM(i_1, i_2)	i_1 – Min(i_1, i_2)	Integer/integer
IDINT(d)	Conversion	Double/integer
IFIX(x)	Conversion	Real/integer
INT(x)	Truncation	Real/integer
ISIGN(i_1, i_2)	Sign i_2 times i_1	Integer/integer
MASK(i)	Form i bit mask, left adjusted	Integer/octal
MAX0(i_1, i_2, ...)	Maximum argument	Integer/integer
MAXI(x_1, x_2, ...)	Maximum argument	Real/integer
MIN0(i_1, i_2, ...)	Minimum argument	Integer/integer
MINI(x_1, x_2, ...)	Minimum argument	Real/real
MOD(i_1, i_2)	i_1 modulo i_2	Integer/integer
OR(x_1, ..., x_n)	Logical sum	Single word
REAL(c)	Real part of complex argument	Complex/real
SIGN (x_1, x_2)	Sign x_2 times x_1	Real/real
SHIFT(i_1, i_2)	Shift i_1, i_2 bit positions: left circular if i_2 positive, right with sign extension if i_1 negative	i_1: Single word i_2: Integer
SNGL(d)	Conversion, unrounded	Double/real

External Function		Argument/Function
ACOS(x)	Arccosine	Real/real
ALOG(x)	Log base e	Real/real
ALOG10(x)	Log base 10	Real/real
ASIN(x)	Arcsine	Real/real
ATAN(x)	Arctangent	Real/real
ATAN2(x_1, x_2)	Arctangent x_1/x_2	Real/real
CABS(c)	Modulus	Complex/real
CCOS(c)	Complex cosine	Complex/complex
CEXP(c)	Complex exponent	Complex/complex
CLOG(c)	Complex log	Complex/complex
COS(x)	Cosine	Real/real
CSIN(x)	Complex sine	Complex/complex
CSQRT(c)	Complex square root	Complex/complex
DATAN(d)	Double arctangent	Double/double
DATAN2(d_1, d_2)	Double arctangent d_1/d_2	Double/double
DATE(x)	Date	Octal/Hollerith
LOCF(x)	Address of argument x	Symbolic/integer
SECOND(x)	CPU seconds	Real/real
SIN(x)	Sine	Real/real

External Function		Argument/Function
SQRT(x)	Square root	Real/real
TAN(x)	Tangent	Real/real
TANH(x)	Hyperbolic tangent	Real/real
TIME(x)	Time of day	Octal/Hollerith
UNIT(i)	Buffer unit i status –1 = Ready, no error 0 = EOF, last operation 1 = Parity error	Integer/real
EOF(i)	0 = No EOF	Integer/real
LEGVAR(x)	Variable characteristic –1 = Indefinite 0 = Out of range 1 = Normal	Real/integer
IOCHEC	Non-buffer parity 0 = No parity error	Integer/integer
LENGTH(i)	Number of words read on previous I/O request on unit i after BUFFER IN	Integer/integer
DATE(x)	Current date	Dummy/Hollerith
DCOS(d)	Double cosine	Double/double
DEXP(d)	Double exponent	Double/double
DLOG(d)	Double natural log	Double/double
DLOG10(d)	Double log base 10	Double/double

External Function		Argument/Function
DSIN(d)	Double sine	Double/double
DSQRT(d)	Double square root	Double/double
EXP(x)	e to the xth power	Real/real
RANF(x)	Random number generator	Dummy/real

STATEMENT FUNCTION

$f(a_1, a_2, \ldots, a_n) = e$

f Function name

a Formal parameters

e Expression

Statement functions are compiled in line.

```
F(P1,P2) = P1*P2+P1/P2+CON
    .
    .
    .
A = F(A,3.0)+B
```

is equivalent to writing:

```
A = (A*3.0+A/3.0+CON)+B
```

LIBRARY SUBROUTINES

DISPLA (nHname,x) $n \leq 10$; x may be real or integer	Display name and value in dayfile
EXIT	Terminate execution
RANGET(x)	Current value RANF

RANSET(x)	Initial value RANF
SECOND(x)	CP time used
REMARK(nHmessage) message $\leq$ 40 characters	Dayfile message
SLITE(i)	Sense light on
SLITET(i,j) i Sense light j Condition: 1 On 0 Off	Sense light test
SSWTCH(i , j) i Sense switch j Condition: 1 Down - on 2 Up - off	Sense switch test
OPENMS(u,ix,L,p)	Mass storage I/O
ERRSET(a , b)	Set maximum number of errors allowed in input data before fatal termination
READMS(u,fwa,n,i)	Read mass storage
WRITMS(u,fwa,n,i)	Write mass storage
STINDX(u,ix,L)	Store index

u Logical unit number

ix First word address of index in central memory

L Length of index

$L \geq 2$ (number of entries) + 1 for name index

$L \geq$ number of entries + 1 for number index

P Type of index

1 for name index

0 for number index

fwa First word address of record in central memory

n Number of central memory words to transfer

i Record number or location of record name

a Error count

b Maximum number of errors in input data

SUBPROGRAM STATEMENTS

Key:

s Symbolic name

f_i Filename

f Function name

a Formal/actual parameters

b Formal/actual returns list parameters

v Subroutine or function name

PROGRAM s

PROGRAM s ($f_1, f_2, \ldots, f_n$)

SUBROUTINE s

SUBROUTINE s ($a_1, a_2, \ldots, a_n$)

SUBROUTINE s, RETURNS ($b_1, b_2, \ldots b_m$)

SUBROUTINE s ($a_1, a_2, \ldots, a_n$), RETURNS ($b_1, b_2, \ldots, b_m$)

FUNCTION f($a_1, a_2, \ldots, a_n$)

REAL FUNCTION f ($a_1, a_2, \ldots, a_n$)

DOUBLE FUNCTION f ($a_1, a_2, \ldots, a_n$)

COMPLEX FUNCTION f ($a_1, a_2, \ldots, a_n$)

INTEGER FUNCTION f ($a_1, a_2, \ldots, a_n$)

LOGICAL FUNCTION f ($a_1, a_2, \ldots, a_n$)

DOUBLE PRECISION FUNCTION f ($a_1, a_2, \ldots, a_n$)

ENTRY s

BLOCK DATA

BLOCK DATA s

EXTERNAL $v_1, v_2, \ldots v_p$

Inter-Subroutine Statements

CALL s

CALL s ($a_1, a_2, \ldots, a_m$)

CALL s, RETURNS ($b_1, b_2, \ldots, b_m$)

CALL s, ($a_1, a_2, \ldots, a_n$), RETURNS ($b_1, b_2, \ldots, b_m$)

RETURN

RETURN b

7000 DIFFERENCES

ECS	Refers to large core memory (LCM) where ECS variables will reside.
READEC WRITEC	Operate on large core memory and small core storage (SCM) in the same manner as 6000 ECS and central memory.
Mass Storage I/O	A 30-bit index is used for random file processing instead of the 24-bit index of the 6000 series.
7000 File Buffers	If no parameter is specified, a buffer size of 412 octal is assumed.
Segment Feature	Not supported for version 1.
Vertical Spacing	For many characters, including A, B, 2, and minus, determined by input/output station serving 7000 SCOPE version 1.
ERRSET (a,b)	Library subprogram added. It sets the maximum number of errors, b, allowed in input data before fatal termination. Error count is kept in a.
Labeled Tapes	Not supported for version 1.

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

SALES OFFICES AND SERVICE CENTERS
IN MAJOR CITIES THROUGHOUT THE WORLD

Pub. No. 60305900 A

Litho in U.S.A.