October 1975

A publication of
Philips-Electrologica B.V.
Marketing Group Small Computers
P.O.Box 245, Apeldoorn – The Netherlands

5122 991 11681

Printed in The Netherlands

PHILIPS

P800M

FORTRAN Reference Data

Data Systems

CONTENTS

LINE FORMAT

Statement Label Field: (card columns 1-5): Contains from 1 to 5 digits.

Line Continuation Field (card column 6): Contains any character other than blank or zero; denotes continuation of a statement from previous line.

Statement Field (card column 7-72): Contains any arithmetic, control, specification, I/O or function statement.

Identification Field (card columns 73-80): Contains card sequence identification; this field is ignored by the compiler and may be left blank if wished.

Comment Line: C in column 1 of any line indicates a comment line; it has no effect upon the program.

Initial Line: The first line of a statement, contains a zero or blank in column 6 and a statement label or blanks in columns 1-5.

DATA TYPES

Integer: occupies 1 word (16 bits). Range $-32767 \leqslant i \leqslant +32767$ $(2^{15}-1)$.

Real: represented in floating point format of two-word mantissa, one word exponent. Range $-2^{2^{15}-1} \leqslant$ real value $\leqslant +2^{2^{15}-1}$ $(|r| \leqslant 10^{9868})$. Accuracy: 8 or 9 decimal digits.

Double precision: occupies 4 words (64 bits); 46-bit mantissa followed by a 16-bit exponent. Range: as for real but with an accuracy of 12 to 13 decimal digits.

Complex: occupies 6 words of memory and is formed by two real numbers.

Logical: occupies 1 word. TRUE has an internal value of −1, FALSE 0.

Hollerith: written as a string of ASCII characters (2 characters to each word).

CONSTANTS

A constant is an explicit numeric value which cannot be redefined.

VARIABLES

A variable is represented by a symbolic name consisting of up to six alphanumeric characters which represents a quantity which may be defined and redefined several times in a program. The first character of the variable name must be a letter.

EXPRESSIONS

An **arithmetic expression** is formed with the arithmetic operators +,−,*,/ and ** and with arithmetic elements (which may be mixed mode).
Results of mixed mode arithmetic expressions:

(For +, −, /, * operations)

1st. operand \ 2nd. operand	I	R	D	C
I	I	R	D	C
R	R	R	D	C
D	D	D	D	X
C	C	C	X	C

(For ** operations)

1st. operand \ 2nd. operand	I	R	D	C
I	I	R	D	X
R	R	R	D	X
D	D	D	D	X
C	C	X	X	X

where: I = Integer (or Logical)
R = Real
D = Double precision
X = Prohibited

A **relational expression** consists of two non-complex arithmetic expressions separated by one of the relational operators .LT., .LE., .EQ., .NE., .GT., .GE.

A **logical expression** is formed with logical elements and the logical operators .OR., .XOR., .AND., .NOT

All operators are subject to a priority order, and expressions are evaluated according to these rules of precedence (unless parentheses are used to change the order).

operator	priority
**	1st (highest)
* /	2nd
+ −	3rd
.LT. .LE. .EQ. .NE. .GE. .GT.	4th
.NOT.	5th
.AND.	6th
.OR.	7th
.XOR.	8th

ASSIGNMENT STATEMENTS

Arithmetic assignment: v=e

v is a variable or array element identifier of any type other than logical; e is an arithmetic expression.

Logical assignment: v=e

v is a logical variable or logical array element; e is a logical expression. Any integer-valued expression may be used instead of a logical expression, and vice-versa.

GO TO assignment: ASSIGN k TO i

k is a statement label; i is an integer variable.

Rules for assignment-arithmetic expressions

If v is	and c is	assignment rule is
Integer	Integer	Assign
Integer	Real	Fix and Assign
Integer	Double precision	Fix and Assign
Integer	Complex	Combination not permitted
Real	Integer	Float and Assign
Real	Real	Assign
Real	Double precision	DP Evaluate and Real Assign
Real	Complex	Combination not permitted
Double precision	Integer	DP Float and Assign
Double precision	Real	DP Evaluate and Assign
Double precision	Double precision	Assign
Double precision	Complex	Combination not permitted
Complex	Integer	Combination not permitted
Complex	Real	Combination not permitted
Complex	Double precision	Combination not permitted
Complex	Complex	Assign

SPECIFICATION STATEMENTS

DIMENSION $v_1(i_1)$, $v_2(i_2)$,..., $v_n(i_n)$
each v(i) is an array declarator; v is the declarator name, i is the subscript which is composed of up to three expressions. The values of the subscript expressions give the maximum size of each dimension.

COMMON /x_1/a_1/ .../x_n/a_n
each x is an optional common block name which, if specified, must not be the same as any variable or array name; each a is a list of variable names, array names or array declarators. All the variables named in a COMMON statement are assigned to storage locations in the order in which the names appear in the statement. If an array is declared in a DIMENSION statement, the subscript need not also be given in a COMMON statement (if the array is to be in common). Memory locations are shared between subprogram and main program variables.

EQUIVALENCE (k_1), (k_2),..., (k_n)
each k is a list of two or more variables or array elements; each element in a list is assigned the same area of memory.

EXTERNAL a_1 a_2..., a_n
each a is a subprogram name. This statement is used to declare subroutine or function names which are used as arguments to another subroutine or function.

<Type>x_1, x_2,..., x_n
each x is a variable name, array declarator or function name which is declared as being of a particular data type (INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL).

DATA list 1/d_1/, list 2/d_2/,.../, list n/d_n/
each list contains the name of variables and array elements that are to be given values; the d's are corresponding lists of optionally signed constant values. If one value is to be assigned to successive variables in the list, that constant may be preceded by an integer constant specifying the repeat number, and an asterisk. An implied DO is accepted in DATA statements provided that all the parameters are integer constants.

CONTROL STATEMENTS

Unconditional GO TO: GO TO n

n is a statement label.

Assigned GO TO: GO TO i (k_1,k_2,...,k_n)

i is an integer variable, the k's are statement labels, one of which must have been assigned to i.

Computed GO TO: GO TO (k_1,k_2,...,k_n),i

the k's are statement labels, i is a non-subscripted integer variable, the value of which determines which statement whose labels is in the list is to be executed next.

Arithmetic IF: IF (e) k_1,k_2,k_3

e is a non-complex arithmetic expression, the k's are statement labels. If the value of e is negative, k_1 will be executed next, if e is zero, k_2 will be executed next, and if e is positive, the next statement executed will be k_3.

Logical IF: IF (e) S

e is a logical or relational expression and S is any executable statement (except a DO or another logical IF).

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

n is a statement label, i is an integer variable (control variable), m_1 (the initial value), m_2 (the terminal value), and m_3 (the incrementation value) are each an integer constant or a non-subscripted integer variable reference (of which only m_3 **must** be greater than zero).

CONTINUE statement: CONTINUE

causes the normal sequence of program execution to be continued.

CALL statement: CALL s(a_1,a_2,...,a_n) or **CALL s**

s is a subroutine name, the a's are actual arguments.

RETURN statement: RETURN

control is returned to a calling program from a function subprogram or subroutine.

STOP statement: STOP or **STOP n**

n is a string of up to four alphanumeric characters* typed out on the operator's typewriter after execution of the statement. Execution of the object program is terminated (compilation is not affected).

PAUSE statement: PAUSE or **PAUSE n**

n is a string of up to four alphanumeric characters, indicating at which point in the program the halt is occurring, and typed out on the operator's typewriter after execution of the statement. Execution can be resumed by typing LF CR.

END statement: END

indicates the physical end of a program unit.

* *'=' character not allowed*

Data Initialization Statement **DATA list$_1$/d$_1$/,list$_2$/d$_2$/,.../,list$_n$/d$_n$/**

'list' contains variables and array element names, d is a corresponding list of constant values. This statement is used to set variables or array elements to initial values.

SUBPROGRAMS

Statement Functions

f(a$_1$,a$_2$,...,a$_n$)=e

f is the function name, the a's are dummy arguments, and e is any arithmetic expression. This statement is defined within the program unit in which it is used.

Function Subprograms

The first statement of an external function must be

FUNCTION f(a$_1$,a$_2$,...,a$_n$)

f is the symbolic name of the function, the a's are dummy arguments. A function is an independent program unit. The body of this type of subprogram must contain an assignment statement assigning a value to the function name, and a RETURN statement which returns the computed value of the function and control, to the calling unit.

Subroutines

A subroutine must begin with the statement

SUBROUTINE x (a$_1$,a$_2$,...,a$_n$) or **SUBROUTINE x**

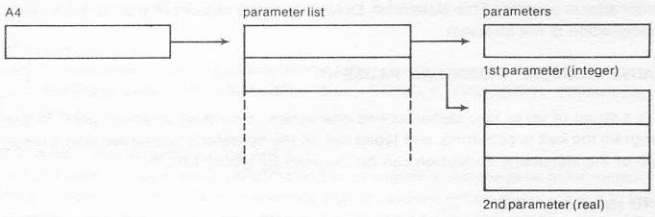
x is the symbolic name of the subroutine, the a's are dummy arguments. A subroutine is an independent program unit, but has no mode associated with its name. A subroutine may contain any statements other than FUNCTION, BLOCK DATA or another SUBROUTINE statement, and it must contain at least one RETURN statement (to return control to the calling program) and an END statement.

Assembly Language Subprograms

Subroutine or Function subprograms may be written in Assembly Language, and the resultant object modules link-edited or link-loaded with FORTRAN modules. The Asembly language subprogram must contain an entry point whose name is in the subprogram name.
At execution time, the subprogram call is interpreted as a

CF A14, X

instruction, branching to the entry point, X.
When this instruction is executed, the A4 register contains the address of the first word of the parameter list which itself contains the address of the first actual parameter; the second word contains the address of the second parameter, and so on.



When execution of the subprogram is completed, control is returned to the calling program by the instruction

RTN A14

The subprogram cannot use the stack (which is defined by the calling program) or A14 (which is the stack pointer, indicating the first free location on the stack) in any other way than as shown.

If a CF instruction is used in the subprogram, the stack must then be defined by the programmer and the contents of A14 saved.

If the subprogram is a Function, returning a computed value to the main program, an integer or logical value will be returned to A1, a real value to A1 -A3 (mantissa in A1, A2; exponent in A3), a double precision value to A1 - A4, and a complex value to A1 - A6 (real part in A1 - A3, imaginary part in A4 - A6).

Block Data

The first statement of the subprogram is

BLOCK DATA

This subprogram is used to enter initial values into common blocks, and contains only EQUIVALENCE, DIMENSION, COMMON, DATA and type statements.

INPUT/OUTPUT STATEMENTS

formatted:	READ (u,f)k WRITE (u,f)k	or or	READ (u,f) WRITE (u,f)
direct formatted:	READ (u's,f)k WRITE (u's,f)k	or or	READ (u's,f) WRITE (u's,f)
unformatted:	READ (u)k WRITE (u)k	or	READ (u)
direct unformatted:	READ (u's)k WRITE (u's)k	or	READ (u's)k

Auxiliary I/O statements:

ENDFILE (u)
REWIND (u)
BACKSPACE (u)

In each case, u is a file code assigned to an I/O device, f is the statement label of a FORMAT statement, k is a list of variables, array names and array elements; and in direct access, s is a sector pointer which indicates a sector number of file u.

STANDARD FILE CODES

(For Basic Executive Monitor)	(For Disc Operating System)
01 Source input	01 Operator's typewriter
02 Listing output	02 Print unit
03 Punch output	03 Punch unit
04 Object code input	10-207 For user assignment
05 Operator's typewriter	> 207 Reserved
06 ASR tape reader	
07 ASR tape punch	
08 Paper tape reader	
09 Paper tape punch	
16 to 255 (max) available for user assignment	

FORMAT STATEMENTS AND FIELD DESCRIPTORS

FORMAT ($q_1 t_1 z_1 t_2 z_2 \ldots t_n z_n q_2$)

q is a series of slashes, or is empty; t is a field descriptor or group of field descriptors; z is a field separator.

Field Descriptors

rIw	Integer conversion
srFw.d	Real conversion (single precision floating point with optional decimal exponent)
srEw.d	Real conversion (single precision floating point with/without exponent)
srGw.d	Real conversion (with or without exponent depending on F-type or E-type evaluation)
srDw.d	Double precision floating point conversion
rLw	Logical conversion
rAw	Alphanumeric character input/output
nX	Insertion of blanks
$nHh_1h_2h_3\ldots h_n$	Hollerith

r is an optional repeat factor which is a positive, unsigned integer constant indicating the number of times to repeat the specified descriptor; s is an optional scale factor designator; w and n are positive, unsigned integer constants representing the total field width of the external character string; d represents the number of positions in the field occupied by the fractional part of the external character string; the letters I, F, E, G, D, L, A, X and H indicate the type of conversion, each h is one ASCII character. (In place of the nH descriptor, Hollerith output data may be represented by a string of characters enclosed in apostrophes.)

COMPILER CONTROL STATEMENTS

IDENT␣m

m is an identifier. This statement **must** be used to assign a name to any module (module = main program, subroutine function or Block-Data subprogram), and is written on the first line of the program module.

OPTIONS i

i is a list of options; these are specified by the letters L, X and/or D which are associated with three Boolean variables in the compiler's work storage. When an IDENT statement is processed these variables are set to zero. When an OPTIONS statement is processed, the values of those Booleans whose names appear in the OPTIONS list are reversed.

L - suppress listing

If this Boolean is set to 1 no source listing will be produced.

X - conditional compilation

If X is 0, any source line whose first character is X will be ignored by the compiler; if X is 1, the line will be compiled as normal.

D - dynamic allocation

If D is 0, the output object module will not be re-entrant. If D is 1, the work storage for the output object module is obtained at run time by means of a dynamic memory allocation request to the Monitor.

JOB CONTROL

When the compiler is stored on disc, a job control command must be given to call the compiler and the source program modules into memory. (Source programs must be loaded onto disc from input peripherals before being compiled.) To call the compiler and to compile a source program from the temporary file /S, or from the user's library, the following command must be given:

FOR␣[/S|<name>][.NL]

Either /S or <name> must be specified. /S indicates that the source program is to be compiled from this file; <name> is the program module identifier and indicates that the source program is in the user's library.
If NL is specified, no listing of the compiled program will be provided.

ERROR CODES

Code	*Meaning*
0000	Invalid statement number
0001	'TO' missing in an ASSIGN statement
0002	Variable in an ASSIGN statement is not integer
0003	Invalid character ending an ASSIGN statement
0004	FORMAT label instead of a statement label in an ASSIGN statement
0100	Left part of an assignment statement invalid
0101	Invalid character ending an assignment statement
0102	Formal argument name given as a statement function name
0103	More than 32 statement functions
0104	More than 8 arguments in a statement function
0105	A statement function's name is used as one of its arguments
0106	Right parenthesis following last argument of a statement function definition is missing
0107	The = sign in a statement function definition is missing
0108	Invalid character ending a statement function definition
0109	Statement function definition not allowed in Block Data subprogram
0200	CALL must be followed by a subroutine name
0201	Name following CALL is already specified as other than a subroutine name
0202	Name following CALL is already specified as a FUNCTION name
0203	Invalid character ending a CALL statement
0300	The first character of a FORTRAN statement must be a letter
0301	Unclassified statement
0600	Illegal variable or array name in DATA
0601	Illegal delimeter in DATA
0602	Formal argument in DATA
0603	Illegal variable or array type inside BLOCK DATA
0604	Common variable outside BLOCK DATA
0605	Illegal subscript expression in DATA
0606	Subscript overflow in DATA
0607	Illegal control variable in a DO-implied Loop
0608	Control variable in a DO-implied loop does not correspond to any subscript
0609	Illegal parameter in a DO-implied loop
060A	Terminal value less than initial value in a DO-implied loop
060B	Increment of zero in a DO-implied loop
060C	Illegal variable list in DATA statement
060D	Zero repeat factor in DATA constant list
060E	Constant list does not correspond to variable list in DATA statement
060F	Illegal constant in DATA statement
0701	Invalid character in a declaration statement
0800	Control variable not specified or not integer in a DO statement
0801	An = sign must follow the DO-loop control variable
0803	Illegal number of parameters in a DO statement
0804	Too many commas in a DO statement

Code	Meaning
0900	A DO statement may not end another DO statement
0901	A DO statement may not be the second part of a logical IF statement
0902	DO must be followed by a statement number
0903	The statement label specifying the end of the DO must not be a FORMAT label
0904	The end of DO-loop statement label not defined
0905	DO ends in an invalid character
0A00	An EQUIVALENCE group declaration must begin with a left parenthesis
0A01	Only actual array/variable may be specified in an EQUIVALENCE statement
0A02	Any subscript element must be an integer constant in an EQUIVALENCE statement
0A03	EQUIVALENCE group declaration must end with right parenthesis
0B00	A name which is already specified in a DIMENSION/EXTERNAL/SUBROUTINE/FUNCTION statement appears in an EXTERNAL statement
0B01	A name already specified in an EQUIVALENCE statement appears in an EXTERNAL statement
0B02	A name already specified in a COMMON statement appears in an EXTERNAL statement
0B03	Unexpected character in an EXTERNAL statement
0C01	Non-executable statement in a module other than Block Data
0C02	Undefined label(s)
0C03	Incomplete DO-loop
0C04	Dynamic allocation with DATA initialization
0C05	Executable statement in Block Data
0C06	No DATA initialization in Block Data
0C07	RETURN missing in subprogram
0E00	Unexpected character ending an auxiliary I/O statement
1000	Number of arguments specified in two uses of the same subprogram is not the same
1100	FORMAT label missing in a FORMAT statement
1101	A FORMAT statement ends a DO-loop
1102	FORMAT label already defined as label of another FORMAT or as a statement label
1103	FORMAT label already referenced as a statement label
1104	'FORMAT' must be followed by (
1105	) is missing at end of FORMAT statement
1106	A FORMAT may not be the second part of a logical IF statement
1400	Integer variable or constant requested and not found
1500	A name was requested and not found
1600	Non-FORTRAN character
1900	A COMMON block name must not be used as a variable name
1901	A digit must follow the decimal point of a constant which has no integer part specified
1902	The character following the E or D exponent must be a + or - or a digit
1903	The exponent part must be less than 32767
1904	The real/imaginary part of a complex constant must not be either logical or integer
1905	) must follow the imaginary part of a complex constant
1906	Relational operator, logical operator or logical constant is incorrectly written (terminal period missing or insufficient letters)
1907	A relational operator/logical operator/logical constant name was not defined

Code	Meaning
1908	A Hollerith constant written nH.. implies n>0
1909	" does not appear within quote marks
190A	A complex constant is not correct: real number missing
190B	Too many numerical constants in this program unit
190C	Hexadecimal constant overflow or $ not followed by a digit or a letter from A to F
190D	Real constant overflow
190E	Too many digits in a real constant
1A00	DO-loop ends in a GO TO statement
1A01	Unexpected character following GO TO
1A02	) missing in computed or assigned GO TO
1A03	Comma missing in computed or assigned GO TO
1A04	Integer variable name not found in computed or assigned GO TO
1A05	Name in computed or assigned GO TO is not a variable name
1A06	Variable in a computed or assigned GO TO is not integer
1A07	Unexpected character ending a GO TO statement
1A08	More than one statement label reference in a simple GO TO
1A09	Illegal character ending a GO TO
1F00	Parenthesis error: IF must be followed by a (with a corresponding)
1F01	Invalid IF expression; such an expression may not contain an =
1F02	An arithmetic IF may not be used to end a DO-loop
1F03	More or less than three statement numbers specified in an arithmetic IF
1F04	Unexpected character ending an arithmetic IF statement
1F05	The control expression of a logical IF statement is neither integer nor logical
1F06	A logical IF statement may not follow another one
2000	Parenthesis error in an I/O statement
2001	Invalid FORMAT label reference in an I/O statement
2002	Unexpected FORMAT reference (neither an array name nor a Hollerith constant)
2003	Format reference is a name but not an array name
2004	No list and no format specified in a WRITE statement
2005	Incorrect variable in an I/O list (not a name nor an array)
2006	Unexpected character in an I/O list
2007	Incomplete DO-implied loop
2008	Invalid array subscript in an I/O list
2300	Invalid statement number
2301	Executable statement not allowed in Block Data
2400	Statement number already defined (as FORMAT or other statement number)
2600	A FORMAT label may not be specified in a list of statement labels
2800	A FORMAT or a statement number may not be zero
2C00	Unexpected character ending an OPTIONS statement
3200	A PAUSE or STOP statement may not end a DO-loop
3500	A RETURN statement may not be used in a main program
3501	A RETURN statement may not end a DO-loop
3601	A RETURN statement may not end a DO-loop
3700	DATA statement has been incorrectly processed
3B00	The requirements of local (non-common) variables and arrays exceed 16384 16-bit words
3B01	Number of dimensions declared in a DIMENSION statement does not correspond with those specified in EQUIVALENCE

Code	Meaning
3B02	Overflow in a COMMON block (more than 16384 words)
3B03	Inconsistency in declaration of groups of Equivalenced names, making allocation impossible
3B04	An array name which is specified in an EQUIVALENCE statement must be declared in a DIMENSION statement
3B05	Two COMMON variables may not be related in EQUIVALENCE
3B06	EQUIVALENCE declaration extends COMMON block backwards
3C00	Invalid sequence for current statement
3E00	Invalid sequence for SUBROUTINE or FUNCTION statement
3E01	A FUNCTION declaration has no argument specified
3E02	Invalid argument in a SUBROUTINE or FUNCTION statement
3E03	Argument in a SUBROUTINE or FUNCTION statement is duplicated
3E04	Illegal delimeter in a SUBROUTINE or FUNCTION statement
3F00	Misplaced common block name in COMMON statement
3F01	Illegal variable dimension in array declaration
3F02	Number misplaced in declaration
3F03	Illegal declaration (general)
3F04	Illegal delimeter in declaration
3F05	Illegal common block name
3F06	Slash missing in COMMON statement
3F07	Illegal array name in array declaration
3F08	Illegal number of dimensions in array declaration
3F09	Formal argument in COMMON statement
3F0A	Common element defined twice
3F0B	Inconsistent variable or array type
3F0C	Dimension overflow in array declaration
4200	Operator incorrect in an arithmetic expression
4201	Hollerith constant in an arithmetic expression
4202	Incorrect character in an arithmetic expression
4203	Erroneous Function call
4204	No argument referenced in an intrinsic function call
4205	Error in arithmetic expression
4206	Hollerith constant in an arithmetic expression or impermissible type mixing
4207	Incorrect type in a logical expression
4208	Error in MIN or MAX function
4209	Subscript is not integer or there are more than 3 subscripts in an array element reference
420A	Subroutine reference in an arithmetic expression
420B	Argument type or number of arguments incorrect in an intrinsic function reference
420C	Wrong number of arguments in a statement function
7F00	This is not an error core but indicates that the END statement has been successfully processed - 'end of compilation' message
C00F	Core overflow. Not enough core available to compile this module.

RUN-TIME ERRORS

Code	Meaning
01	No more core storage can be allocated for a re-entrant module, or for an I/O operation
02	Wrongly generated object code - ask software maintenance
03	Incorrect value given for index variable in a GO TO statement
04	Negative step value in a DO-loop
10	Overflow in integer arithmetic operation
11	Undefined result for ISIGN function
12	Overflow or undefined result in integer exponentiation
13	Arithmetic overflow in subscript computation or subscript not positive
20	Overflow in real addition or subtraction
21	Underflow in real addition or subtraction
22	Overflow in real multiplication or division
23	Underflow in real multiplication or division
24	Real division by zero
25	Overflow in real negation or in ABS or SIGN computation
26	Undefined result for SIGN function
27	Overflow in IFIX function
28	Undefined result in real exponentiation
29	Overflow in real exponentiation
2A	Undefined result for ALOG function
2B	Overflow in ALOG function
2C	Negative SQRT argument
2D	Overflow in EXP function
2E	Undefined result for raising a real to an integer power
2F	Overflow in raising a real to an integer power
30	Overflow in double precision addition or subtraction
31	Underflow in double precision addition or subtraction
32	Underflow in double precision negation
33	Undefined result for DSIGN function
34	Overflow in double precision multiplication
35	Underflow in double precision multiplication
36	Overflow in double precision division
37	Underflow in double precision division
38	Double precision division by zero
39	Negative argument to DSQRT
3A	Undefined result for ALOG10 function
3B	Overflow in DEXP function
3C	Negative argument to DLOG function
3D	Negative argument to DLOG10 function
3E	Undefined result for ATAN2 function
3F	Undefined result for DATAN2 function
40	Undefined result in double precision exponentiation
41	Overflow in double precision exponentiation
42	Undefined result when raising a double precision to an integer power
43	Overflow in raising a double precision to an integer power
44	Second argument to MOD function is zero
45	Second argument to AMOD function is zero
46	Erroneous or un-normalized argument in real or double precision operation or function
47	Second argument to DMOD function is zero
48	Overflow in CONJG function
49	Overflow in DIM function

I/O ERRORS

Code	Meaning
70	Irrecoverable I/O error during an auxiliary I/O operation
71	Irrecoverable I/O error during a READ or WRITE operation, or illegal file code for a random I/O request
72	Illegal FORMAT specification
73	Field width too small or zero
74	Group or field repeat count is zero
75	Error in a string of ASCII characters between quotes in a format: Such a format is not allowed with a READ statement, or End or format statement encountered before last quotation mark.
76	No conversion specified in format for next I/O list element
77	Maximum number of characters allowed for a physical record on the specified unit is exceeded (I/O buffer overflow)
78	First parenthesis of format specification is missing (when format reference is an array name)
79	Type of variable is not compatible with field descriptor
7A	More than ten levels of parentheses in a format specification
7B	Illegal logical variable (Input): The first non-blank character is neither T nor F; or the whole external field is blank
80	Illegal input character (possibly a decimal point or exponent in an integer)
81	Overflow during input conversion
82	Illegal unformatted record
83	Logical (unformatted) record is too small
84	Unformatted READ is not allowed on typewriter
85	Only one sector (400 characters) is allowed for an unformatted disc I/O.

FULL FORTRAN TRANSCODER

The Full FORTRAN Transcoder translates Full FORTRAN object modules into machine code instructions.

Control Commands

C	copy the current module untranscoded onto the output file
D	delete the current module
E	end the current transcoding session
:EOF	punch out an :EOF (End Of File) mark onto the output file, and end the current transcoding session (not for disc Transcoder).
T[,NL][,NP]	transcode the current module. NL and NP are optional parameters. If NL is specified, no listing of the transcoded module will be produced (control messages and error messages are always displayed on the ASR typewriter), if NP is specified, no punched object code will be produced.

Each command must be terminated by CR LF.

Control Messages

IDENT<name>	identification of the current module
:EOS	end of segment mark read
:EOF	end of file mark read
T:	a control command is requested
COMPILED LENGTH=xxxx TRANSCODED LENGTH=yyyy	length of the compiled and the transcoded module, expressed as a hexadecimal number of characters

Error Messages

IDT.MIS.	identification statement missing or invalid
END.MIS.	end/start cluster missing
CLS.ERR.	erroneous cluster type
COM.MOD.	the current module is not a compiled one
BLK.DAT.	block data modules cannot be transcoded
TBL.OVF.	table overflow
MOD.ERR.	unsuccessfully compiled module
DYN.ALL.	dynamic allocation forbidden
IO ERROR xxxx yyyy	unrecoverable I/O error, xxxx=file code, yyyy=returned status
OBJECT CODE NOT USABLE	because of a fatal error the transcoded object code is not usable

P800M High Speed Fortran Compiler

The information given for P800M Full FORTRAN also applies to P800M High Speed FORTRAN, except for:
- I/O statements
- compiler control statements
- job control
- error messages.

INPUT/OUTPUT STATEMENTS

Sequential I/O statements

READ (u[,f] [,ERR=a] [,END=b)[k]
WRITE (u[,f] [,ERR=a])[k]

u is the file code, f a FORMAT statement label, an array name or a Hollerith string specifying the format, a is a label to which a branch is made if an irrecoverable I/O error occurs, b is a label to which a branch is made if an End-of-File mark is read, and k is an I/O list. The parameters between [ ] brackets are optional, and the order of the ERR and END parameters may be reversed.

Implicit I/O statements

PRINT f[,ERR=a],k	is equivalent to **WRITE (2,f[,ERR=a])k**
PUNCH f[,ERR=a],k	is equivalent to **WRITE (3,f[,ERR=a])k**
READ f[,ERR=a] [,END=b],k	is equivalent to **READ (SEI,f[,ERR=a] [,END=b])k**

Disc random access

DEFINE FILE u_1(m_1,r_1,f_1,v_1),u_2 (m_2,r_2,f_2,v_2),. . .

must be used to define a file before a random I/O statement refers to the file. The parameter u specifies the file code, m is the number of records, r is the record length (in words or characters, depending on the value of f), f specifies formatted or unformatted access (E = formatted, record size in characters;

U = unformatted, record size in words; L = formatted or unformatted, record size in characters), and v is an integer variable, which contains the number of the next available record after a READ or WRITE, and the number specified in the FIND statement after a FIND operation.
The DEFINE FILE statement can only be used in main programs, for subprograms with random I/O operations the file must be defined in the main program, and the variable v must be transferred to the subprogram.

CALL CLOSE (u)
releases the buffer and working storage (holding information about the file u), which was obtained dynamically after the first reference to file u.

FIND (u's)

starts a disc seek operation to find the record s on the file u.

READ (u'sⱡ,fⱢ ⱡ,ERR=aⱢ)k
WRITE (u'sⱡ,fⱢ ⱡ,ERR=aⱢ)k

are the direct access READ and WRITE statements. The parameter u specifies the file code, s is the record number, f specifies the format, a is the label of a statement to which a branch is made if an irrecoverable I/O error occurs, and k is an I/O list. The parameters between ⱡ Ⱡ brackets are optional.

CALL CREATE (u)

writes dummy records on the file u, so that the whole file, and not only the used part, is kept when the file is catalogued. As the routine destroys part of the file, it must be called before any information is written on the file.

COMPILER CONTROL STATEMENTS

IDENT ␣ m

m is an identifier. The IDENT statement must be the first one in a module.

OPTIONS ␣ ⱡXⱢ ⱡ,MⱢ ⱡ,CⱢ ⱡ,DⱢ

X – conditional compilation
Any source line with an X in column 1 is compiled only if X has been specified in the OPTIONS statement.

M – printing of a map
If M is specified, a map is printed with all the variable names and label numbers of the module.

C – object code listing
If C is specified, the generated object code is printed in assembly language format.

D – run-time diagnosis
If D is specified, the run-time error codes are extended with messages which locate the incorrect statement.

If an OPTIONS statement is used in a module, it must be the first one after the IDENT statement.

JOB CONTROL

The compiler is called, and compilation is started, by the control statement

HSF ␣ ⌊/S⌋ <name>⌋ ⱡ,NLⱢ

One of the parameters between ⌊ ⌋ brackets must be specified. /S indicates that the source module is to be compiled from the /S file, <name> is the module identifier, and indicates that the source module is in the user's library.
The optional parameter NL suppresses the listing of the source code.

ERROR MESSAGES

Errors detected during compilation are printed in the source code listing. The message consists of 'ERROR', followed by a sentence explaining the error.

The run-time and I/O error codes are the same as for P800M Full FORTRAN. The following error codes have been added:

Code	*Meaning*
86	Random file not declared in a DEFINE FILE statement
87	File overflow in a random I/O operation
88	Formatted I/O not allowed with U specified in the DEFINE FILE statement
89	Record number 0 not allowed for a random file
8A	Unformatted I/O not allowed with E specified in the DEFINE FILE statement
8B	This random file is not a disc file

REAL TIME FORTRAN LIBRARY ROUTINES

Name/arguments	*Subroutine/ function*	*Subroutine function, or function value*
ACTIV(i,j,k,m,)	S	Activate program i*), build a 2-word ECB with beginning address j, and transmit parameters through parameter block with address k. If k=0, no parameters are transmitted. Status of request returned in m: m=1 request accepted m=2 program i not connected to a level m=3 program i unknown m=4 no more core available to record request
AFDEV(i,j,k,p,l)	S	Assign file code i to the physical device k**) with device address l, and build a 6-word assign block with beginning address j. Status returned in p: p=1 assignment completed p=2 I/O error on disc p=3 no spare entry in file code table p=4 no free file description table p=5 device unknown p=6 disc overflow p=7 file unknown
AFEQU(i,j,k,p)	S	Assign file code i to device or file already having file code k. Build a 6-word assign block with beginning address j. Status returned in p: as in AFDEV, and p=8 second file code unknown p=9 more than 7 file codes assigned to the same disc file
AFPERM(i,j,k,p,n)	S	Assign file code i to the permanent disc file n*) on the disc with file code k. A 6-word assign block is built with beginning address j. Status returned in p: as in AFEQU, except for p=8

Name/arguments	Subroutine/ function	Subroutine function, or function value
AFTEMP(i,j,d,p,m)	S	Assign file code i to a temporary file on disc with file code d. If m>0, m granules are allocated to the temporary file, if m=0 the file is sequential. A 6-word assign block is build with beginning address j. Status returned in p: as in AFPERM.
ATTACH(i,j)	S	Reserve device or disc file with file code i exclusively for the calling program. Status returned in j: j=TRUE device or disc file attached already j=FALSE request accepted
ATTCHW(i,j)	S	Reserve device or disc file with file code i exclusively for the calling program, and put the program in wait state if device or disc file has been attached already. Status returned in j: j=TRUE device or disc file attached already j=FALSE request accepted
BCLR(j,k)	S	Reset bit k***) of word j
BSET(j,k)	S	Set bit k***) of word j
BTEST(j,k)	F	TRUE if bit k***) of word j=1 FALSE if bit k***) of word j=0
CLEV(i,j,m)	S	Connect program i*) to software level j. Status returned in m: m=1 connection accomplished m=2 program connected already m=3 level <50 or >63, or program unknown m=4 no more core available to record request (DRTM systems only)
CTIM(i,j,k,l,m)	S	Connect program i*) to timer j, with pulse rate k (0≤k≤2047). Wait l timer cycles before first activation. Status returned in m: m=1 connection accomplished m=2 program connected already m=3 program or timer does not exist m=4 wrong parameter m=5 space for parameter block cannot be allocated

Amount of time associated with timer number:

	Standard clock	Special clock
j=0	20 msec	< 20 msec
j=1	100 msec	20 msec
j=2	1 sec	100 msec
j=3	1 min	1 sec
j=4	1 hour	1 min
j=5	-	1 hour

CTIMC(i,j,k,t,m)	S	Connect program i*) to timer j, with pulse rate k (0≤k ≤127). Activate program at absolute time t (t is array, t(1)=hours, t(2)=minutes, t(3)= seconds). Status returned in m: see CTIM. For timer number values, see CTIM.

Name/arguments	Subroutine/ function	Subroutine function, or function value
DATE(j,m)	S	Place date and time of day into first 6 elements of integer array j (month-day-year-hours-minutes-seconds). Status returned in m: m=1 request accepted m=2 request not accepted
DECB(i,j,k,l)	S	Build an Event Control Block with beginning address j, $80 in the event byte, file code value i, buffer address k, and requested length l.

j	$80	i	event byte/file c.
j+2	k		buffer address
j+4	l		req. length
j+6	0		eff. length
j+8	0		status
j+10	0		sector number

DELETE(i,l)	S	Delete file code i from the Monitor File Code Table. If i is a temporary disc file, the granules of the file are released. Status returned in l: 1=FALSE deletion completed 1=TRUE file code deleted, but I/O error during release of granules of temporary disc file
DETACH(i,j)	S	Detach device or disc file with file code i from the calling program. Status returned in j: j=FALSE device or disc file detached j=TRUE device or disc file does not exist, or has been attached to another program
DLEV(i,j,m)	S	Disconnect program i*) from software level j. Status returned in m: m=1 disconnection accomplished m=2 program busy m=3 program does not exist
DTIM(i,j,m)	S	Disconnect program i*) from timer j Status returned in m: m=1 disconnection accomplished m=2 timer unknown, or program not connected to that timer m=3 program unknown
GETBUF(b,l,s)	S	Allocate a storage area of l bytes to the calling program, return beginning address in b, set s to FALSE if storage area has been allocated, or to TRUE if allocation is not possible.
GETBFW(b,l)	S	Allocate a storage area of l bytes to the calling program, return beginning address in b. Put program in wait state if the storage area is not available.
GETCOM(i,k)	S	Inform the monitor that the calling program is waiting for message k**), to update the event byte in the Event Control Block with beginning address i.
IADDR(m)	F	Address of variable m or first element of array m
IAND(m,n)	F	m.AND.n
IBTEST(j,k)	F	-1 if bit k***) of word j=1, 0 if bit k=0
IBYTE(b,d)	F	Value of byte with address b+d (base address b+d bytes)
IEOR(m,n)	F	m.XOR.n
IOR(m,n)	F	m.OR.n

Name/arguments	Subroutine/ function	Subroutine function, or function value
ISHFT(m,n)	F	Word m shifted right n positions if n<0 Word m shifted left n positions if n>0 Word m if n=0 0 if |n| >15
IWORD(b,d)	F	Value of word with address b+2d (base address b+d words)
LADD(m,n)	F	Address m + address n
LCMP(m,n)	F	−1 if address m<address n 0 if address m = address n +1 if address m>address n
LEVENT(j)	F	Test event byte with address j Value of function = TRUE: event byte set FALSE: event byte not set
LSUB(m,n)	F	Address m − address n
MOVTOH(b_1,b_2,1)	S	Move storage area of 1 bytes with beginning address b_1 to area with beginning address b_2, starting with the highest address
MOVTOL(b_1,b_2,1)	S	as MOVTOH, but starting with the lowest address
NOT(m)	F	Logical complement (.NOT.m)
POST(j)	S	Update event byte with address j (set bit 0 to 1)
RANDOM(x)	F	A random real number between 0 and 1. X is a dummy argument.
RELBUF(b)	S	Release storage area with beginning address b
RIO(i,j,k)	S	Start an I/O operation on a random access disc file, and return control to the calling program as soon as the I/O operation has been initialized. i is the function code ($OA=read, $OB=write), j is the ECB address, and k the sector number.
RIOW(i,j,k,m)	S	Start an I/O operation on a random access disc file, and wait for completion. i is the function code (see RIO), j the ECB address, and k the sector number. The status of the I/O operation is returned in m.
RVALUE(b,d)	F	Value of the real variable with address b + 6d (base address + d real values)
SIO(i,j)	S	Start a sequential I/O operation with function code i, ECB address j, and return control to the calling program as soon as the I/O operation has been initialized. The status of the request is returned in m. Function code values: i=1 basic read i=2 standard read i=5 basic write i=6 standard write i=9 replace record i=20 or $14 skip forward up to EOS mark i=22 or $16 skip forward up to EOF mark i=34 or $22 write EOF mark i=38 or $26 write EOS mark i=48 or $30 get information about a file code i=49 or $31 rewind i=51 or $33 skip backward one block i=52 or $34 skip forward one block
SIOW(i,j,m)	S	Start a sequential I/O operation with function code i and ECB address j, and wait for completion. The I/O status is returned in m. For function code values see SIO.

Name/arguments	Subroutine/ function	Subroutine function, or function value
START(i,j,k,m)	S	Start program i*) after a delay of j timer units of timer k (for timer numbers see CTIM). Status returned in m: m=1 request accepted m=2 program already connected to a timer m=3 program name unknown m=4 wrong parameter m=5 space for a parameter block cannot be allocated
STBYTE(b,d,v)	S	Store the 8 rightmost bits of word v at address b+d (base address b+d bytes)
STREAL(b,d,v)	S	Store real value v at address formed by truncating b+6d to an even value
STWORD(b,d,v)	S	Store word v at address b+2d (base address b+d words), truncated to an even value
SWLEV(j)	S	Switch to next program at software level j (or to program at current level + 1 if j=0)
TIME(j,m)	S	Place time of day into first 6 elements of array j (hours-minutes-seconds-tenths of seconds- fiftieths of seconds- number of pulses special clock, 0 for standard clock) Status returned in m: m=1 request accepted m=2 request not accepted
TRNON(i,t,m)	S	Start program i*) at absolute time t (t is array: t(1)=hours,t(2)=minutes,t(3)=seconds) Status returned in m: m=1 connection to timer accomplished m=2 program already connected to a timer m=3 program or timer does not exist m=4 wrong parameter m=5 space for a parameter block cannot be allocated
WAIT(j,k,m)	S	Stop the running program during j units of timer k (for timer numbers see CTIM). Status returned in m: m=1 request accepted m=2 request not accepted
WECB(j)	S	Stop the running program, and wait for setting of event byte with address j.

*) i is 6-character Hollerith constant, variable, array element or array containing program name.

**) a 2-character Hollerith constant, a variable or array element.

***) k= $\log_2$ of the value of the bit position: least significant bit is bit 0.

EXTENSIONS TO, AND RESTRICTIONS ON AMERICAN STANDARD FORTRAN

EXTENSIONS

Line Syntax
There is no limit imposed on the number of continuation lines.

The first character of any line may be the letter X which is used in the conditional compilation feature.

A comment line may be followed by a continuation line.

Subscripts
Any integer-valued expression is accepted as a subscript expression.

Arithmetic Expressions
Mixed mode arithmetic expressions are allowed.

DO Statement
The control variable, the initial, terminal and incrementation values may be redefined during execution of the range of the DO.
Only the incrementation value must be greater than zero.

Format Specifications
The FORMAT statement reference in a READ or WRITE statement may be specified by a label, an array name or a Hollerith constant which constitutes a valid format specification. When the format specification is in an array, an nH descriptor may form part of the specification.

Hollerith Descriptor
On output, the nH descriptor may be replaced by a string of characters enclosed within apostrophes. Data are then transferred exactly as written.

Control Statements
Two special statements - IDENT and OPTIONS - are available which control the compilation process.

Logical and Integer Expressions
Any integer-valued arithmetic expression may be used instead of a logical expression in any statement or expression, and conversely, any logical expression may be used instead of an integer expression.

RESTRICTIONS

In a statement function or assignment statement the = sign must be written in the initial line of the statement.

In any control statement the keyword (GOTO, CONTINUE etc.) must be written in the initial line of the statement; no part of it may be carried on to a continuation line.

The comma following the initial parameter in a DO statement must appear in the initial line of the statement.

Both STOP and PAUSE statements may be followed by a string of not more than four alphanumeric characters (as opposed to five in A.S.A.).

The number of subscript expressions must not differ from the number of dimensions declared for an array. However, this does not prevent a two- or three-dimensional array from being made equivalent to a one-dimensional array (by application of the element successor rule).