PDP-1 COMPUTER
ELECTRICAL ENGINEERING DEPARTMENT
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CAMBRIDGE, MASSACHUSETTS 02139

PDP-48

FORTRAN

November 8, 1971

Preface

This memo is not intended to be of tutorial value. It presumes a general knowledge of FORTRAN, such as can be obtained from A FORTRAN IV PRIMER by E. I. Organick.

Definitions and notation

For clarity, examples of FORTRAN source statements will be written in upper case.

Procedure means function or subroutine. (External procedures are sometimes called subprograms.) A function returns a value, and is called by a reference in an arithmetic expression. A subroutine does not return a value, and is called by the CALL statement.

Mode (also known as type) is one of integer, real, logical, complex, and doubleprecision. With very few exceptions, all data processed by FORTRAN programs are in one of these modes. (One exception is a procedure name argument to a procedure.) The mode of a constant is determined by its form. The mode of a variable is determined by explicit mode declarations, implicit mode declarations, or the IJKLMN rule, in that order. The mode of the value returned by a function is determined in the same way, except for intrinsic and library functions.

The IJKLMN rule is that variables and functions whose names begin with I, J, K, L, M, or N are integer mode, and others are real. This rule may be changed by an IMPLICIT statement.

A dimension declaration is a declaration consisting of the array name followed by a list, in parentheses, of the maximum values of each subscript. It may appear in DIMENSION, LOGICAL, REAL, DOUBLEPRECISION, INTEGER, COMPLEX, and COMMON statements.

A data declaration is a declaration specifying the initial contents of a variable or array, and may appear in DATA, DIMENSION, LOGICAL, REAL, DOUBLEPRECISION, INTEGER, and COMPLEX statements.

An ASF is an arithmetic statement function (internal function defined in a single statement which looks like an arithmetic assignment statement with subscripted left side).

Source program format

A statement is a string of characters up to a carriage return, not counting carriage returns hidden by centerdot or in character strings. A centerdot causes all characters up to and including the next carriage return to be ignored, and is used for continuing statements on more than one line. There is no limit to the number of continuation lines that a statement may have. An empty statement or statement with C, period, or slash as the first character is ignored. Centerdot may not be used to continue comments. Except in character strings, space and tab are ignored. There are no other restrictions on format or line position. Specifically, statement numbers and the rest of the statement do not have to occupy specific "fields". Statements may begin at the left margin with no tab, but beware of CONTINUE, COMMON, COMPLEX, CALL, and other statements beginning with C. These must be preceded by a space or tab if they have no statement number, to prevent their being treated as comments.

Letters may be in upper or lower case. Brackets may be used in place of parentheses. The case of letters is ignored, i.e., the names ABC, ABc, and abc are one and the same.

The first non-empty line of a program is the title. It is printed on the typewriter at the beginning of compilation and is not a statement.

Character strings may be written with "numberH" notation or may be enclosed in single quotes. In the former case, case shifts are counted as characters, and they are not filtered. In the latter case, case shifts are filtered, and the string is assumed to begin and end in lower case. Single quotes may be placed in the second kind of string by using a pair of consecutive single quotes. All characters are legal in either kind of string. Strings in other than format statements are left justified if they contain more than 3 characters. They are right justified and considered to be integer constants if they contain 3 or fewer characters.

Arithmetic expressions

A symbol is a string of up to 6 letters or digits, the first of which is a letter. A symbol may be used as a variable, array name, function name, subroutine name, COMMON block name, or NAMELIST name. Except for COMMON block names, a symbol may be only one of the above unless it is a dummy symbol for an internal procedure, in which case it may have one meaning within the procedure and another meaning elsewhere.

A variable is a name for a register in which one data item may be stored during execution. The mode of the variable determines the type of data that may be stored.

An integer datum is a signed integer with magnitude not greater than 131071 decimal or 377777 octal.

A real datum is an approximation to a real number. It has magnitude between 10↑-75 and 10↑70, or is zero. 26 bits (about 7 1/2 decimal digits) of precision are kept. It is stored in two consecutive words.

A doubleprecision datum has magnitude between 10↑-39000 and 10↑39000, or is zero. 69 bits (about 21 decimal digits) of accuracy are kept. It is stored in 5 consecutive words.

A logical datum has the value .TRUE. (stored as 1) or .FALSE. (stored as 0).

A complex datum is a pair of real data, representing the real and imaginary parts of a complex number. It is stored in four consecutive words.

The mode of a variable is determined by an explicit mode declaration (INTEGER, REAL, LOGICAL, COMPLEX, or DOUBLEPRECISION statements), or, if none is present, from its first letter, using an IMPLICIT declaration or the IJKLMN rule.

An array is a set of variables named by a single symbol. Individual items are selected by a string of subscripts. Each array must appear in a dimension declaration, giving the number of subscripts and the maximum value of each. Any reasonable number of subscripts may be used. Each subscript may vary between 1 and the limit given in the dimension declaration. The items of an array (each of which is 1, 2, 4, or 5 words long depending on the mode) are stored in ascending locations in memory, the first subscript "varying" most rapidly.

example - DIMENSION JJ(2,3)
The elements of the array, in order of increasing address, are
JJ(1,1) JJ(2,1) JJ(1,2) JJ(2,2) JJ(1,3) JJ(2,3)

An array reference is an array name followed by a parenthesized list of subscripts. Subscripts may be of arbitrary complexity. The number of subscripts must match the number of dimensions given in the declaration, or must be one. In the latter case, it is a "linear subscript" and selects an item from the array as a linear list.

The mode of an array is the mode of each item in the array and is determined in the same way as the mode of a variable.

Constants

An integer constant is an optionally signed number, octal number, or character string of 3 or fewer characters. An octal number is a number followed by the letter O. A character string is enclosed in single quotes or is preceded by a number, giving the length of the string, and the letter H. The characters are right justified and filled with zero on the left.

examples - 123 -123o 1ha 'foo'

The letter O in an octal constant may be in front of the digits in a data declaration, where the constant cannot be confused with a symbol.

A real constant is a number with a decimal point, an exponent preceded by E, or both. The exponent may be signed, as may the entire constant. If there is no exponent, the number of digits must be ≤ 9 to distinguish it from a doubleprecision constant.

A doubleprecision constant is a number with a decimal point, an exponent preceded by the letter D, or both. The exponent may be signed, as may the entire constant. If there is no exponent, the number of digits must be > 9.

A logical constant is .TRUE. or 1B for truth, .FALSE. or 0B for falsehood. the forms T and F are also permitted in a data declaration, where the constant cannot be confused with a symbol.

A complex constant is a pair of real constants separated by a comma and enclosed in parentheses. Each part may be signed, as may the entire constant.

Note - - The form left paren, real part, comma, imaginary part, right paren, is legal only for constants. (A,B+C) is illegal. Use CMPLX(A,B+C) instead.

A long character string constant is a string of more than 3 characters. It is not one of the 5 data types, and is legal only in data declarations, procedure arguments, and IO lists. In a data declaration, it is treated as a list of

integers. In a procedure argument it is treated as an array. A long character string is left justified and filled with zeroes on the right.

A statement label constant is a statement number followed by the letter S. It is not one of the 5 data types, and is legal only in data declarations and subroutine arguments. In a data declaration it is used to initialize a variable to be used in an assigned GO TO. In a subroutine argument it is used to specify an address for an error return. The letter S may precede the number in a data declaration, where it cannot be confused with a symbol.

A function reference is a function name followed by a parenthesized argument list. The mode of a function reference is the mode of the function name, which, except for intrinsic functions and standard library functions, is determined in the same way as the mode of a variable.

An expression is a constant, variable, array reference, function reference, expression enclosed in parentheses, expression preceded by a unary operator, or expressions separated by binary operators.

The arithmetic operators are listed below

operator(s)	acceptable mode(s)	
.AND.	L	logical and
.OR.	L	logical or
.THEN.	L	implies (B or not A)
.EXOR.	L	logical exclusive or
.EQV.	L	logical equivalence
.NOT. (unary)	L	logical negation
∧ .A.	I	bitwise and
∨ .V.	I	bitwise or
⁻ .EV.	I	bitwise exclusive or
.N. (unary)	I	ones complement
.REM.	I	remainder
.LS.	I	I shifted left by J
.RS.	I	I shifted right by J
.LR.	I	I rotated left by J
.RR.	I	I rotated right by J
+	I, R, D, C	
- (unary or binary)	I, R, D, C	
×	I, R, D, C	
/	I, R, D, C	
↑ .P. ××	I, R, D, C	exponentiate
.ABS. (unary)	I, R, D	absolute value
.GT. .G. >	I, R, D	
.GE. ≥	I, R, D	
.EQ. .E.	I, R, D, C	
.NE.	I, R, D, C	
.LT. .L. <	I, R, D	
.LE. ≤	I, R, D	

Where parentheses are not used to direct the order of computation, operations are performed in the order shown below.

.ABS. unary - (done first)
.LS. .LR. .RS. .RR.
.N.
.A. ∧
.EV. ⁻
.V. ∨
×× ↑ .P.
/ .REM.
×
+ binary -
.GT. .G. > .GE. ≥ .EQ. .E. .NE. .LT. .L. < .LE. ≤
.NOT.
.AND.
.EXOR.
.OR.
.THEN.
.EQV. (done last)

Where two or more non-commutative operators of the same priority level are used, the operations are performed from left to right except for exponentiation, which is performed from right to left.

All arguments to an operator are first converted to an acceptable mode by the smallest possible subset of the following conversions.

doubleprecision → real
The less significant bits of the fraction are discarded.

real → doubleprecision
Zeros are appended to the fraction.

doubleprecision → integer
The number is truncated toward zero. It is not rounded.
2.718281828459O → 2 -3.1415926536 → -3

integer → doubleprecision
The fraction is zero.

complex → real
The real part is used.

real → complex
Zero is used for the imaginary part.

real → integer
The number is truncated toward zero. It is not rounded.
2.999 → 2 -2.999 → -2

integer → real
The fraction is zero.

integer → logical
Zero is converted to .FALSE. Anything else is converted to .TRUE.

logical → integer
.FALSE. becomes 0. .TRUE. becomes 1.

Except for exponentiation with integer second argument, all arguments are then converted to a consistent mode according to the following 5 steps.

If there are any doubleprecision arguments, all integer arguments are converted to doubleprecision.

If there are any real or complex arguments, all integer arguments are converted to real.

If there are any complex arguments, all doubleprecision arguments are converted to real.

If there are any doubleprecision arguments, all real

arguments are converted to doubleprecision.

If there are any complex arguments, all real arguments are converted to complex.

Whenever two or more arguments undergo the same conversion, the arithmetic operation is performed on them first, and the result converted. For example A+I+B+J+C+K is rearranged as A+B+C+FLOAT(I+J+K). The integer arguments are added as integers and then converted.

When the second argument for exponentiation is integer, it is not converted to make it consistent with the first, that is, the operations doubleprecision↑integer, real↑integer, and complex↑integer are performed without conversion.

Except for connectives (.GT., .EQ., etc.) and exponentiation, the mode of the result is the mode of the arguments. The result of a connective is always logical. The result of exponentiation has the mode of the first argument.

Except in IO lists and arguments to procedures, expressions are always converted to the required mode. For example, the right side of an arithmetic assignment statement is converted to the mode of the left side, a computed GO TO index is converted to integer, and the test expression in a logical IF is converted to logical. There is no "mixed mode" error.

When the result of an integer operation is not integral (this may happen in division or exponentiation with negative exponent) it is truncated toward zero. In the forms real↑real and doubleprecision↑doubleprecision, the base must be non-negative. 0↑0 is undefined in all modes.

Executable statements

Statement labels (statement numbers)

Any executable statement may be preceded by a label, which is a decimal integer between 1 and 99999. Statement labels are used to generate addresses for GO TO and IF statements, and to indicate ends of DO loops. Labels are ignored on non-executable statements other than FORMAT. Labels on FORMAT statements must not conflict with labels on executable statements.

Arithmetic assignment

The arithmetic assignment statement consists of a variable name or array reference, an equals sign, and an expression. The expression is evaluated and stored in the indicated variable or array element. If the left side is an array reference, its subscripts may be of arbitrary complexity and follow the rules for array references in expressions. The expression to the right of the equals sign is converted to the mode of the variable or array on the left by the smallest possible subset of the conversions given in the section on expressions.

Simple GO TO

The statement GO TO followed by a number causes transfer of control to the numbered statement.

Computed GO TO

The statement GO TO followed by a parenthesized list of statement numbers (separated by commas) followed by a comma followed by an expression, is a computed GO TO. Control transfers to the statement number whose position in the list is the value of the expression. The expression may be of arbitrary complexity. If it is not integer, it is converted to integer. If the value of the expression is 1, control is transferred to the first statement in the list, etc.

example - GO TO (10,30,25), J+K/2

ASSIGN

The statement ASSIGN, statement number, TO, variable or array reference, stores a pointer to the specified statement in the variable or array element. The variable or array should be integer. This statement is used in conjunction with the assigned GO TO statement.

example - ASSIGN 4 TO J(K+L)

Assigned GO TO

The statement GO TO followed by a symbol or array reference followed by a comma followed by a parenthesized list of statement numbers, causes transfer of control to the statement whose pointer was last stored in the variable or array element by an ASSIGN statement. The parenthesized list of statement numbers is required and must contain all possible statements to which control may be transferred. The compiler uses thes information internally.

example - GO TO J(N+1),(3,4,5,14)

The use of an array for assigned GO TO's facilitates certain "pushdown stack" operations.

Arithmetic IF

The statement IF followed by a parenthesized expression followed by 3 statement numbers separated from each other (but not from the expression) by commas, is an arithmetic IF. The expression is converted to integer, real, or doubleprecision, and tested. Control transfers to the first, second or third statement if the value is strictly negative, zero, or strictly positive, respectively.

example - IF (J-2) 10,15,10

Logical IF

The statement IF followed by a parenthesized expression followed by any executable statement other than DO, is a logical IF. The expression is converted to logical. The statement following the expression is executed if the value is .TRUE.

example - IF (P .OR. Q) J = J+1

DO

A DO loop is begun by the statement DO, a statement number, symbol, equals sign, and 2 or 3 expressions separated by commas. The loop ends just after the terminal statement, whose number is given in the DO statement. The symbol before the equals sign in the DO statement is the index. The expressions after the equals sign are the initial value, test value, and increment, respectively. They may be of arbitrary complexity. If the increment is missing, 1 (integer) is used. When the DO statement is executed, the index is set to the initial value, and subsequent statements up to and including the terminal statement (the "range") are executed repeatedly. The increment is added to the index after each iteration. If the result is less than or equal to the test value, the range is executed again. Otherwise it proceeds to the statement after the terminal statement. The range is always executed at least once. Mode conversions are performed where necessary to do the addition and testing. The testing is done as if by a .LE. connective, so the arguments are converted to real, integer, or doubleprecision. Because of the way the test is performed, the increment should always be positive. The terminal statement may be any executable statement other than GO TO, STOP, RETURN (unless under logical IF), or another DO. It is permissible to jump in and out of DO loops at any time. The last value of the index (after being incremented) is preserved upon normal exit from a DO loop. DO loops may be nested to any reasonable depth.

CONTINUE

The CONTINUE statement does nothing. It is used as the terminal statement of a DO loop in order to be able to go to the end of the range of the loop.

PAUSE and STOP

The statement PAUSE or STOP, followed by nothing, an octal number, or a character string enclosed in single quote marks, causes execution to stop. If the statement has a number or string, it is typed out first. The number, if present, will be printed in octal. The STOP statement terminates execution with the dam instruction. The PAUSE statement waits for any character to be typed in and then proceeds. If a character string is given, it must be enclosed in quote marks. "H" format is not permitted (it would be ambiguous). The same effect as a STOP statement may be achieved by running into an END statement (running off the end of the program or procedure) or by an end of file on a READ statement when no end-of-file address is given.

examples - PAUSE PAUSE 777 PAUSE 'foo'

CALL

The statement CALL follwed by an internal or external subroutine name, followed by an optional parenthesized argument list, calls the subroutine. The form of the arguments is discussed in detail in the section on procedures.

RETURN

The statement RETURN followed by nothing, a number (for non-standard subroutine returns), or a parenthesized expression (for function returns), returns from the innermost procedure. This statement is discussed in detail in the section on procedures.

READ and WRITE

These statements are used to effect transmission of data to or from the outside world. READ or WRITE is followed by a parenthesized list of one or two items, followed by the IO list. No comma precedes the IO list. The first item in parentheses is the unit number, which is an expression of arbitrary complexity. It is converted to integer if necessary. The second item, if present, specifies a FORMAT statement number or array to be used for "variable format". These statements are treated in detail in the section on input/output.

Declarations

The mode declarations (REAL, LOGICAL, COMPLEX, and DOUBLEPRECISION)

These declarations are used to explicitly specify the mode of a variable, dummy symbol, or function name. The mode word is followed by the names to be declared, which are separated from each other, but not from the mode word, by commas. Mode declarations may be placed anywhere, except for declarations on dummy symbols for internal procedures, which must be inside the procedure definition.

If a function is called in a program, a mode declaration must be given unless it is a library or intrinsic function or the correct mode is given by an IMPLICIT declaration or the IJKLMN rule. In a FUNCTION program which defines an external function, the mode must be declared, either with an explicit mode declaration or in the FUNCTION statement itself, unless the correct mode is given by an IMPLICIT declaration or the IJKLMN rule. In a program in which an internal function is defined and used, its mode must be declared either inside or outside of the definition, or in the INTERNAL FUNCTION statement itself, unless the correct mode is given by an IMPLICIT declaration or the IJKLMN rule.

A mode declaration may give dimension information for arrays by replacing the symbol name with a dimension declaration (following the symbol name by a parenthesized list giving the maximum value of each subscript).

example - LOGICAL A, B(4,5,2), C

A mode declaration may specify initial contents of variables or arrays by following the symbol name (or dimension declaration) by a data declaration, which is enclosed in slashes.

example - LOGICAL A/T/,B(4,5,2)/40x.FALSE./,C

The DIMENSION declaration

The DIMENSION declaration specifies the size and subscript function for an array. It consists of the word DIMENSION followed by a list of dimension declarations, separated from each other by commas. Each symbol to be declared is followed by a parenthesized list giving the maximum values of each subscript. Subscripts may vary between 1 (not 0) and the given limit, inclusively. Except in dimension declarations for dummy symbols, the limits must be integer constants, since the compiler needs to know how much space to reserve. In dimension declarations for dummy symbols, the limits may be variables. This feature, called "adjustable dimensions", facilitates coding procedures that manipulate arrays of different sizes. The limits in the dimension declaration are dummy symbols, and the corre-

sponding arguments are numbers by which the calling program specifies the size of the array being transmitted.

example - subroutine to transpose a square matrix

```
      SUBROUTINE TRAN(A, SIZE)
      INTEGER SIZE
      DIMENSION A(SIZE, SIZE)
      DO 13 I = 2, SIZE
      DO 13 J = 1, I-1
      K = A(I, J)
      A(I, J) = A(J, I)
13    A(J, I) = K
      RETURN
```

When this subroutine is called with a square matrix and length of a side as arguments, the subscript function and terminal value for the outer DO loop will be adjusted to the size of the array.

Warning - the subscript limits (except possibly the last) in the dimension declaration for a dummy array must always match the corresponding limits in the original array. Calling the above subroutine with a second argument smaller than the actual size of the array in an attempt to transpose just the upper left corner will not work.

A DIMENSION statement may specify initial contents of an array by following the dimension declaration by a data declaration, which is enclosed in slashes.

example - DIMENSION FOO(5)/.1, .2, .3, .4, .5/

The COMMON declaration

A COMMON declaration specifies that certain variables and arrays are to be placed in named blocks of common storage, where they can be linked by the loader among several programs.

The word COMMON is followed by symbols and block names, the latter enclosed in slashes. The block names are looked at only by the linking loader, and may conflict with other symbols. They must conform to the requirements for FORTRAN symbols.

The symbols to be placed in common storage are separated from each other, but not from the word COMMON or the preceding block name, by commas. A comma separating a block name from the preceding symbol is optional.

example - COMMON /FOO/ A,B, /BAR/ C,D /MUNG/ E,F

Each symbol is assigned to the block whose name most recently precedes it, or, if no block name precedes it in the statement (or the preceding block name is null), it is assigned to a block called "blank common". Within each block, variables and arrays are assigned to increasing addresses in the order in which they appear.

COMMON statements may be placed anywhere. Dummy symbols may not be assigned to common storage. A COMMON statement may give dimension information for arrays by replacing the symbol name by a dimension declaration.

The DATA declaration

The DATA declaration specifies initial contents of variables and arrays. It may be placed anywhere. The word DATA is followed by pairs of variables lists and constants lists, the pairs being separated from each other by commas. In each pair, the variables list is a list of variables, arrays, array elements, or implied DO's, separated from each other by commas. The variables list may not contain dummy symbols. The constants list is enclosed in slashes and consists of constants, or constants with repetition counts, separated by commas. Constants must match the mode of the variables to which they are assigned.

```
examples - DATA A/3.0/,B,C,D/4.0,5.0,6.0/,E,F,G/3*7.0/
           DATA (Q(I),I=1,20)/10*1.0,10*-1.0/
```

An implied DO consists of a left parenthesis, a list of variables, arrays, array elements, or implied DO's, a comma, an index name (which is always considered integer and is unrelated to any other symbol), equals sign, initial value, comma, final value, comma, increment, and right parenthesis. The increment and preceding comma may be omitted, in which case the increment is one. The compiler iterates over the list, letting the index vary in the same way as it would in a DO statement. Implied DO's may be nested to any reasonable depth. Implied DO parameters and array element subscripts are integer expressions that may be of arbitrary complexity. They may not contain references to intrinsic or other functions, and the only variables permitted are indices of implied DO's controlling the expression.

The constants list consists of constants separated by commas. The constants may be preceded by a positive integer and a multiplication sign, to indicate that the constant is to be repeated. Several forms are legal for constants in DATA statements but not in expressions because they would be ambiguous or generate more than one data item.

Long character string constants are legal. They are treated as lists of integer constants, one for each three characters.

T and F may be used for logical constants.

Statement numbers, in either of the forms 123S or S123, may be used for initializing variables for assigned GO TO's.

Octal constants may be written as o77, +o77, -o77, o+77, or o-77.

For each variables list/constants list pair, the compiler scans the two lists in parallel, making the indicated assignments. When either list runs out, the remainder of the other list is ignored.

The EXTERNAL declaration

The EXTERNAL statement declares symbols that appear as arguments to procedures as entry points to external procedures, to prevent the compiler from treating them as variables. This declaration is not needed for symbols that are already recognizable as entry points. The word EXTERNAL is followed by the symbols to be declared, separated from each other by commas. The EXTERNAL statement may be placed anywhere.

The EQUIVALENCE declaration

The EQUIVALENCE statement specifies that certain variables and array elements are to occupy the same locations in memory. It may be used to place symbols in specified areas of common storage, to give special names to certain elements of an array, to patch programs with misspelled symbols, to give a variable different names in different modes, etc. The word EQUIVALENCE is followed by a list of declarations, each enclosed in parentheses and separated from each other by commas. Each declaration contains two or more symbols, arrays, or array elements that are to be assigned to the same location. Subscripts for array elements must be constants. An array name is treated as the first element of the array.

example - EQUIVALENCE (A,B,C),(D(4),E(3,1)),(F(2,4),G)

This declaration may be placed anywhere. It may not be used on dummy symbols.

The IMPLICIT declaration

The IJKLMN rule may be changed by the IMPLICIT statement. The word IMPLICIT is followed by one or more declarations, separated from each other by commas. Each declaration is a mode word (INTEGER, REAL, LOGICAL, COMPLEX, or DOUBLEPRECISION) followed by a parenthesized list of letters and pairs of letters. Single letters mean that symbols beginning with that letter are to have the specified mode, unless overridden by a rule of higher priority. Pairs of letters separated by a minus sign cause the declaration to apply to all letters from the first to the second inclusively.

examples - IMPLICIT REAL(I-N),INTEGER(A-H,O-Z)
IMPLICIT LOGICAL (A-C,F,R-U)

Letters not affected by IMPLICIT statements are treated according to the IJKLMN rule. The IMPLICIT statment may be placed anywhere. Its meaning inside an internal procedure definition is the same as outside.

The FUNCTION, SUBROUTINE, and ENTRY declarations

The first two of these statements begin internal or external procedure definitions. The word FUNCTION or SUBROUTINE is followed by the name of the procedure and the parenthesized dummy symbol list. The word FUNCTION may be preceded by a mode word to specify the mode of the returned value. For an internal procedure, the word INTERNAL precedes everything else. The ENTRY statement is used to specify an alternate entry point to a procedure. ENTRY is followed by the alternate name. No dummy symbol list is given.

The END declaration

Inside an internal procedure definition, the END statement terminates the definition and resumes compilation of the main program. Otherwise, END indicates the end of the source program.

The BLOCK DATA declaration

The statement BLOCK DATA, which must be the first statement in the source program, indicates that the program is a "block data subprogram". Such a program contains no executable statements. It consists of declarations, principally COMMON declarations and DATA declarations initializing variables and arrays in common storage. The block data subprogram is used to construct one or more common blocks to be linked by the loader to programs requiring them.

The NAMELIST declaration

This statement is used to provide format information for IO statements that are to be processed by the namelist interpreter. The word NAMELIST is followed by one or more declarations separated by commas. Each declaration is the name of the list enclosed in slashes followed by a list of variables or array names separated from each other by commas. This statement may be placed anywhere. Dummy symbols may not appear in the list, and the list name may not conflict with any other symbol This statement is treated in greater detail in the section on input/output.

example - NAMELIST /FOO/A,B,C, /BAR/D,E

The FORMAT declaration

This statement provides format information for IO statements that are to be processed by the standard format interpreter. The word FORMAT is followed by a parenthesized list of format specifications and slashes. The specifications need not be separated if there IO no ambiguity, or they may be separated by a comma. If too many consecutive unseparated specifications are used, the "field list overflow" error may occur, which can be corrected by providing commas. This statement must have a statement number, and may be placed anywhere, It is treated in more detail in the section on input/output.

example - 123 FORMAT(2I4L5A3,2L4,F6.3,2E8.3)

An end-of-file address for a READ statement may be specified by placing the statement number (with no S) in front of the IO list as if it were the first element. If an end of file is encountered, control will transfer to the specified statement. If no end-of-file address is specified, execution terminates.

example - READ (1,123) 50, I, J, K
will go to statement 50 on end of file

The IO list

The IO list consists of one or more arrays, expressions, sublists, or implied DO's separated by commas. In a WRITE statement the expressions may be of arbitrary complexity. In a READ statement, only variables, array elements, and arrays should be used. In either case, IO list elements must be of the correct mode (the mode given in the format statement). No mode conversion is performed. Complex data must be transmitted with two format specifications for real data (E, F, or G format). An IO list item should not contain a function call that results in the execution of another IO statement.

An implied DO is a left parenthesis, one or more IO list items (including implied DO's), a comma, index name, equals sign, initial value, comma, test value, comma, increment, and right parenthesis. If the increment and preceding comma are omitted, one (integer) is used. The initial value, test value, and increment may be of arbitrary complexity. The items under control of the implied DO are processed repeatedly, with the index varying exactly as it would in an ordinary DO loop. A sublist is one or more IO list items enclosed in parentheses. Implied DO's and sublists may be nested to any reasonable depth. An array name is treated as if it were in an implied DO iterating over its linear subscript from beginning to end.

example -
```
12  FORMAT ('a table of square roots'//100(I5F10.7/))
    WRITE (0,123) (J, SQRT(FLOAT(J)), J = 1,100)
```

The unit number

A unit number of zero refers to the typewriter. Other unit numbers refer to files.

Format statements and arrays

If the unit number is followed by a number, it refers to a FORMAT statement. If it is followed by an array name, the "variable format" interpreter treats the text in the array as if it were a format statement. If no number or array is specified, the data are transmitted in unformatted "binary" form. The IO list and format statement are scanned in parallel until the IO list is exhausted. If the end of the format statement is reached, it repeats from the last left parenthesis.

There are 10 types of basic format specifications. All except X and H may be preceded by a repetition count.

Dw.d - Doubleprecision. W is the field width, and d is the number of digits after the decimal point. The present format interpreter ignores both. This specification (with its repetition count, if any) may be preceded by a scale factor, which is an optionally signed integer followed by the letter P.

Ew.d - Real (or one part of complex datum). Similar to D.

Fw.d - Real. Similar to D.

Gw.d - "General" (real). Similar to D. The present format interpreter does not distinguish between E, F, and G.

Iw - Integer (decimal). W is the field width. The present format interpreter ignores it.

Ow - Octal integer. Similar to I.

Lw - Logical. Similar to I.

Aw - Character. W is the number of characters. They are packed in an array 3 per word.

nX - Generate n spaces in output. It is ignored in input. No data item is transmitted.

nHstring or 'string' - "Hollerith". The string is written from or read into the format statement. No data item is transmitted.

A slash generates a carriage return in output. It is ignored in input. No data item is transmitted.

Format specifications may be grouped within parentheses. A number preceding the left parenthesis will cause the

entire group to be repeated that number of times.

Output format

Data items written under control of D, E, F, G, I, O, and L fields are separated from each other on the same line by tabs. A, X, and H fields are not separated. L fields are written as t or f.

Input format

Leading spaces, tabs, and carriage returns are ignored when reading D, E, F, G, I, O, and L data items. Numbers must be followed by a space, tab, or carriage return. Plus signs and other extraneous characters are not permitted. Exponential notation may not be used. L fields are scanned until a t or f is found.

Functions and subroutines

Functions and subroutines (collectively called procedures) are of three kinds - internal procedures, external procedures, and intrinsic functions. Internal procedures are those defined in the program in which they are used. An internal procedure definition is an arithmetic statement function (ASF) or a section of source program beginning with the INTERNAL FUNCTION or INTERNAL SUBROUTINE statement and ending with the END statement. An external procedure is supplied by a library, either the standard FORTRAN library or a library of procedures written by the user. In the latter case the procedures may be written in machine language or as FUNCTION or SUBROUTINE programs in FORTRAN. Intrinsic functions are simply arithmetic operators written in function-like notation.

Because internal and external procedures have many features in common, these features will be described together.

Function and subroutine calls

Subroutines (internal and external) are called with the CALL statement. CALL is followed by the name of the subroutine and an optional parenthesized argument list. Functions are called by a "function reference", which is the name of the function followed by a parenthesized argument list, in an arithmetic expression. The argument list for a function is required. The compiler recognizes a function reference by the existence of a symbol, which does not appear in a dimension declaration, followed by a parenthesized list.

Function and subroutine arguments

An argument to an internal or external procedure may be

(1) Any arithmetic expression. The corresponding dummy symbol must be of the same mode. If the expression is a variable or array element, the procedure may modify it (use it on left side of arithmetic assignment statement, use it in a READ statement, use it as a DO index, etc.), and the corresponding variable in the calling program will be modified.

(2) An array name. The corresponding dummy symbol must be dimensioned and of the same mode. In order to make the subscript calculations in the procedure match those of the calling program, all dimensions of the dummy symbol except possibly the last must match those of the original array. For example, it is not possible to transmit the upper left corner of a rectangular array by merely using smaller dimensions in the procedure than in the calling program, because the first dimension is used for computing the correspondence between a given pair of subscripts and an actual

position in the array. It is possible, by using as the argument an element of the array other than the first, to effect an offset between the array in the calling program and the dummy array in the procedure.

(3) An external procedure name. Internal procedures and intrinsic function names may not be transmitted as arguments. In the procedure, the dummy symbol is used as the subroutine name in a CALL statement or function name in a function reference. In the calling program the external procedure must be recognized as such by the compiler. Use of the symbol elsewhere in the calling program in a CALL statement or function reference will suffice. If this is not the case, the procedure name must appear in an EXTERNAL declaration.

```
example - EXTERNAL SIN
          CALL FOO(SIN,1,2)
```

If the declaration were not used, the compiler would think SIN is a variable.

(4) A character string of more than 3 characters. The actual argument that is transmitted is an array consisting of the characters packed 3 per word, left justified. (A character string of 2 or fewer characters is an integer constant and is right justified.)

(5) A statement number. This is used for "error exits". It may be used only with subroutines (internal or external), not with functions. The argument in the calling program is the statement number followed by the letter S. In the subroutine the corresponding dummy symbol is the times sign (×). The statement RETURN n where n is an integer and the nth dummy symbol is ×, will return to the statement whose number is the corresponding argument instead of the statement following the CALL.

Declarations on dummy symbols

Dummy symbols in functions and subroutines must match the corresponding arguments. The compiler does not insert mode conversions for procedure arguments, and, except in the case of intrinsic functions, does not check for incorrect correspondence. When an argument is a function name, array name, or arithmetic expression, the corresponding dummy symbol must have the same mode, which may require a mode declaration in the procedure. If the argument is an array name, the dummy symbol must also appear in a dimension declaration. No DATA, COMMON, EQUIVALENCE, NAMELIST, INTERNAL FUNCTION, INTERNAL SUBROUTINE, or ENTRY declarations are permitted for dummy symbols. The EXTERNAL declaration may be used where applicable, but is not necessary.

Return from function or subroutine

Normal return from a subroutine (internal or external) is effected by the RETURN statement with no argument. Execution resumes at the statement following the CALL statement. Return to an alternate point in the program is effected by

RETURN n or RETURN (n)

where n is an integer and the nth dummy symbol is x. The subroutine will return to the statement whose number is the argument corresponding to the x.

```
example - CALL S(L,M+N,123,3)
          .
          .
          INTERNAL SUBROUTINE S(I,J,X,K)
          .
       20 IF(I .GE. 0) RETURN 3
          .
          END
```

If the conditional in statement 20 is satisfied, control will return to statement 123.

A value must be specified when returning from an internal or external function. This may be done either by placing the value in parentheses after the word RETURN or by storing the value in the variable with the same name as the function, and then executing a RETURN with no argument.

```
example - INTERNAL FUNCTION FOO(A,B,C)
          .
          .
          RETURN (A+B)
          .
          FOO = B+C
          RETURN
          .
          END
```

Inside a function (and only inside the function) the name of the function is treated as an ordinary variable. When a RETURN statement without argument is executed, the contents of that variable are used as the function value.

Alternate entry points

Alternate entry points to internal or external procedures may be specified by means of the ENTRY statement. A procedure may be called by the name given in the FUNCTION or SUBROUTINE statement or by any name given in an ENTRY statement. In the latter case execution will begin just after the ENTRY statement.

```
example - INTERNAL FUNCTION SIN(X)
          Y=X
          GO TO 10
          ENTRY COS
        Y=1.57079633-X
    c if the variable y were not used, and COS stored
    c its modified argument back into x, the argument
    c in the calling program would be clobbered
```

No dummy symbol list is given with the ENTRY statement. The dummy symbol list given in the FUNCTION or SUBROUTINE statement applies to all entries. All entry points to a function must have the same mode. Any entry point name may be used for storing the value to be returned by a function.

Internal functions and subroutines

Internal functions and subroutines are compiled as part of the program in which they are used, without the need to be linked by the loader. They have the further advantage that they may use any variables in the main program, without the need for them to be transmitted through a COMMON block. They include as a special case the "arithmetic statement function", the only type of internal procedure available in most FORTRAN systems.

An internal procedure definition other than an ASF is a section of program beginning with the INTERNAL FUNCTION or INTERNAL SUBROUTINE statement and ending with the next END statement. An explicit mode declaration of an internal function may be made immediately following the word INTERNAL.

```
examples - INTERNAL SUBROUTINE SUBR
           INTERNAL FUNCTION FUN(A,B)
           INTERNAL LOGICAL FUNCTION LLL(J,K)
```

Internal procedure definitions, including ASF definitions, may be placed anywhere in the source program except within another internal procedure definition. If an internal procedure definition is placed within the main program, control will pass over it. DO loops must be nested properly within each internal procedure. It is possible to jump out of an internal procedure. Jumping into a procedure is not allowed. An internal procedure name may not be used as an argument to any procedure.

Mode of internal function

The mode of the value returned by an internal function may be specified by a mode word in the INTERNAL FUNCTION statement, by an explicit mode declaration (which may be inside or outside of the function definition), by an IMPLICIT declaration, or by the IJKLMN rule. All entry points must be of the same mode.

Dummy symbols for internal procedures

Dummy symbols in internal procedures are not related to symbols of the same name in the main program or other procedures. Hence a dummy symbol may have a type or dimension declaration in an internal procedure, and the same symbol may have a different declaration elsewhere.

Arithmetic statement functions

An arithmetic statement function is a special type of internal function defined in one statement, containing the name of the function, the parenthesized dummy symbol list, an equals sign, and an expression giving the value to be returned.

example - FOO(A,B,K) = A+B**J*C(2+K*K)

An arithmetic statement function is recognized by the fact that it appears to be an arithmetic assignment statement with a subscripted left side, but the symbol on the left side does not appear in a dimension declaration. Since an ASF definition cannot contain any declarations, the dummy symbols cannot be dimensioned (although the function may contain references to other arrays). No explicit mode declarations are possible either, so the modes of the dummy symbols will be determined by an IMPLICIT declaration (if any), or by the IJKLMN rule.

External functions and subroutines

Some of the library functions (the "standard" FORTRAN library functions) are known to the compiler. This means that, if the name is used as an external function, the compiler will assume the correct mode for the returned value unless there is an explicit mode declaration to the contrary. No automatic mode conversion of arguments will be performed.

The standard library functions are:

name	n args	arg mode	value mode	description
EXP	1	real	real	exponential
DEXP	1	double	double	double exponential
CEXP	1	complex	complex	complex exponential
ALOG	1	real	real	natural logarithm
DLOG	1	double	double	double logarithm
CLOG	1	complex	complex	complex logarithm
ALOG10	1	real	real	common (base 10) logarithm
DLOG10	1	double	double	double common (base 10) log
CARG	1	complex	real	phase of complex number
CABS	1	complex	real	modulus of complex number
DMOD	2	double	double	A-[A/B]xB
SQRT	1	real	real	square root
DSQRT	1	double	double	double square root
CSQRT	1	complex	complex	complex square root
CBRT	1	real	real	cube root
DCBRT	1	double	double	double cube root
SIN	1	real	real	sine
DSIN	1	double	double	double sine
CSIN	1	complex	complex	complex sine
COS	1	real	real	cosine
DCOS	1	double	double	double cosine
CCOS	1	complex	complex	complex cosine
TAN	1	real	real	tangent
DTAN	1	double	double	double tangent
CTAN	1	complex	complex	complex tangent
COT	1	real	real	cotangent
DCOT	1	double	double	double cotangent
CCOT	1	complex	complex	complex cotangent
SINH	1	real	real	hyperbolic sine
DSINH	1	double	double	double hyperbolic sine
CSINH	1	complex	complex	complex hyperbolic sine
COSH	1	real	real	hyperbolic cosine
DCOSH	1	double	double	double hyperbolic cosine
CCOSH	1	complex	complex	complex hyperbolic cosine
TANH	1	real	real	hyperbolic tangent
DTANH	1	double	double	double hyperbolic tangent
CTANH	1	complex	complex	complex hyperbolic tangent
ASIN	1	real	real	inverse sine
DASIN	1	double	double	double inverse sine
CASIN	1	complex	complex	complex inverse sine
ACOS	1	real	real	inverse cosine
DACOS	1	double	double	double inverse cosine

CACOS	1	complex	complex	complex inverse cosine
ATAN	1	real	real	inverse tangent
DATAN	1	double	double	double inverse tangent
CATAN	1	complex	complex	complex inverse tangent
ATAN2	2	real	real	phase of (B + A1)
DATAN2	2	double	double	double phase of (B + Ai)

The library has many other functions and subroutines. For a complete listing of these routines, see PDP-50, RELOCATABLE SUBROUTINE LIBRARY.

The compiler must know the mode of the value of every external function. Except in the case of one of the standard library functions given above, a mode declaration may be necessary. The mode that the compiler assumes for the value of a function is determined by explicit mode declarations, the known modes of standard library functions, any implicit declaration, or the IJKLMN rule, in that order of priority.

External procedures written in FORTRAN

An external procedure may be written in FORTRAN as a program whose first statement is a FUNCTION or SUBROUTINE statement. The word FUNCTION or SUBROUTINE is followed by a parenthesized dummy symbol list. The list is optional in the case of a subroutine. The mode of the value returned by an external function may be specified by a mode word preceding the word FUNCTION or by a declaration in the program.

```
examples - COMPLEX FUNCTION THETA2(A,B)
           FUNCTION FOO(X)
           SUBROUTINE SORT(A,N)
```

A FUNCTION or SUBROUTINE program ends with the END statement, as does a main program It may contain internal procedures (each with its own END statement).

Intrinsic functions

An intrinsic (sometimes called "built in") function is an arithmetic operation that is used by a construct in the source program that looks like a function call Some intrinsic functions (such as ABS) are translated directly into machine instruct ons. Others, such as SNGL, are translated into one or more calls to library subroutines.

The name of an intrinsic funct on may not be used in another context elsewhere in the program. If, for example, it is used as a variable, it will be a variable throughout the entire program, and its meaning as an intrinsic function will be discarded. Similarly, any dimension, COMMON, EXTERNAL, etc. declaration, or explicit mode declaration not in agreement with the correct mode, will remove the meaning of a symbol as an intrinsic function. It is not necessary to declare the mode of an intrinsic function.

Arguments to intrinsi functions must match the requirements given below for number. Automatic mode conversions of arguments will be performed.

name	n args	arg mode	value mode	description
DFLOAT	1	integer	double	convert
FLOAT	1	integer	real	convert
IFIX	1	real	integer	convert
DBLE	1	real	double	convert
SNGL	1	double	real	convert
REAL	1	complex	real	real part
AIMAG	1	complex	real	imaginary part
AMOD	2	real	real	A-[A/B]×B real remainder
MOD	2	integer	integer	A-[A/B]×B remainder
ABS	1	real	real	|A|
IABS	1	integer	integer	|A|
DABS	1	double	double	|A|
AINT	1	real	real	sign(A)×[|A|] (truncate)
INT	1	real	integer	sign(A)×[|A|] (truncate)
IDINT	1	double	integer	sign(A)×[|A|] (truncate)
SIGN	2	real	real	|A|×sign(B)

ISIGN	2	integer	integer	|A|×sign(B)
DSIGN	2	double	double	|A|×sign(B)
DIM	2	real	real	A-min(A,B)
IDIM	2	integer	integer	A-min(A,B)
CMPLX	2	real	real	A + iB
CONJG	1	complex	complex	complex conjugate
AMAXO	≥2	integer	real	maximum
AMAX1	≥2	real	real	maximum
MAXO	≥2	integer	integer	maximum
MAX1	≥2	real	integer	maximum
DMAX1	≥2	double	double	maximum
AMINO	≥2	integer	real	minimum
AMIN1	≥2	real	real	minimum
MINO	≥2	integer	integer	minimum
MIN1	≥2	real	integer	minimum
DMIN1	≥2	double	double	minimum

Because of automatic mode conversion, the intrinsic functions to perform mode conversion (IFIX, FLOAT, SNGL, DBLE, etc.) are rarely needed. In arguments to external or internal procedures, where automatic mode conversion does not take place, the use of these explicit mode conversion functions may be required. For example, to calculate the (real) square root of the integer J, SQRT(FLOAT(J)) must be used. SQRT(J) would result in an uncorrected mode mismatch.

APPENDIX I

Use of the Compiler and Loader

Simple Compilation

A simple compilation is a compilation of a main program that calls no external procedures which are not stored on the FORTRAN tape (See PDP-50, RELOCATABLE SUBROUTINE LIBRARY, for a list of available subroutines and where they are stored). In order to compile a FORTRAN main program it is necessary to have the microtape containing the FORTRAN compiler mounted on tape drive n. The compilation is begun by typing the command

nF

to ID or to ET (For complete descriptions of these programs, see PDP-23, INVISIBLE DEBUGGER (ID), and PDP-22, EXPENSIVE TYPEWRITER (ET)). The compiler will type the title (first nonempty line) of the program on the typewriter as it begins the compilation. Any errors detected by the compiler will cause informative messages to be printed.

If the compilation fails, control will be returned to ID. If the compilation is successful, the loader will be started automatically. When the loader types "options - ", type a carriage return. If loading is successful, the size of the loaded program will be printed. Otherwise, an error message will be printed. The loaded program will be placed onto drum field 1 and into core. If the loaded program is more than 4096 words long, only the first 4096 words will be placed onto drum field 1. Control is passed to ID.

Execution of the program may be begun by typing

P

or

1020

to ID. If the program crashes, control will be returned to ID which will type a message of the form:

xxxxx↑↑	hlt	(These are ID's standard messages for in-
or		indicating that a halt instruction or an
yyyyy<<	zzzzz	illegal memory reference has occurred)

To stop a running program, hit the call button. This will also return control to ID. From ID, control may be returned to ET by typing

E

or a fresh copy of the program (or the first 4096 words of

the program if the program is longer than 4096 words) may be obtained and execution started by typing:

```
0_        (zero underbar)
101U
1020
```

Compiling a Subroutine or a Function Subprogram

The procedure for compiling a subroutine or a function is the same as that for compiling a main program, except that at the termination of compilation the loader is not started but rather control is passed to ID. If the compilation was successful, the object file can be placed in a library. This is done by saving the object file on any microtape containing a file system (any of the public tapes numbered 1 through 7 will do). The object program can be saved by typing the following sequence of commands:

```
nF              (n is the number of the drive on which the file
                 tape is mounted)
s fortlib progname          (progname is any desired file name
                             (nine or fewer characters))
b
```

(For an explanation of what these commands do, see PDP-42, MICROTAPE FILE SYSTEM). At the end of this sequence ID is in control and the compiled subprogram is stored on the microtape mounted on drive n. The next section explains how to compile a main program which calls subprograms saved on a library tape.

General Compilation

To compile a main program which uses subprograms that are saved on a library tape, proceed as in the case of a simple compilation. At the end of a successful compilation, the loader program will be started. When the loader types "options - ", type the number of the tape drive on which the libary tape is mounted followed by a carriage return. The loader will then search all files on that library tape whose first name is "fortlib" for procedures that have been referenced. One external procedure may reference other external procedures. All will then proceed as in a simple compilation.

Loader

The loader combines relocatable files into binary programs. The operation of the loader may be more closely observed by the use of various options which may be selected when the loader types "options - ". At this point, various characters may be typed. Each character specifies an option. If a character which does not correspond to any option is typed, the loader will type "options - " again.

CHARACTER	OPTION
any digit	The digit is taken to be the unit number of a tape drive containing a library tape.
m	Print a load map giving the name, location, and length of each procedure that is loaded.
r	Print every reference to an external procedure. This option is only effective if the load map option is also selected.

APPENDIX II

Features Not Implemented

Non-standard subroutine returns (i.e. RETURN 3).

BLOCK DATA subroutines.

NAMELIST IO.

Variable format IO.

Entire arrays in IO list.

Complex item in IO list.

IO to devices other than the typewriter through WRITE and READ statements.

Scale factors (P field) in FORMAT statements.

H format input.

Use of an internal function name as a "variable" inside of the internal function definition, in any context other than the following:

```
NAME = expression
RETURN
```

The following intrinsic functions are not implemented:

DIM
IDIM
AMAX1
MAX1
AMIN1
MIN1
DMAX1
DMIN1

APPENDIX III

Known Bugs

doubleprecision ↑ real does not work. The necessary conversion fails to occur.

The use of internal functions in expressions is dangerous. Temporaries used by the statement calling the internal function may be altered during the evaluation of the internal function.

Exceedingly long character string constants (i.e. thousands of characters) will not compile.

FORTRAN coded external procedures (functions or subroutines) will not work correctly if they are loaded into core modules other than 0 and they contain arrays that are neither in common nor parameters to the procedure. The load map option of the loader can be used to detect this condition. This bug may be avoided by putting all arrays occuring in FORTRAN coded subprograms in COMMON blocks.