A REVIEW OF
SOME DIALECTS OF
FORTRAN

ALCOCK SHEARING & PARTNERS

8 IDDESLEIGH HOUSE,
CAXTON STREET,
LONDON S.W.1.

TELEPHONE
01-799 3644

A REVIEW OF
SOME DIALECTS OF
FORTRAN

by David Muxworthy of
The Edinburgh Regional Computing Centre

and Brian Shearing of
Alcock Shearing and Partners

This document reviews various implementations of the language FORTRAN in comparison with the standard dialect published by the American Standards Institute.

First Edition February 1970

Second Edition August 1970

A REVIEW OF SOME DIALECTS OF FORTRAN

PREFACE TO THE SECOND EDITION

In this edition we have corrected some errors found in our first edition. We have also altered the text to reflect some of the changes made to the compilers since our first edition. Changes occur at such a rate, however, that it is impossible for us to be right up to date.

We have been told that a compiler's limitations should not be classed as "differences" between it and the Standard. This is fair comment; we would not wish to discourage manufacturers from publishing the limits their compilers have. But we have retained all this information none the less; it is much too valuable to throw away.

D.T. Muxworthy
B.H. Shearing

August 1970

A REVIEW OF SOME DIALECTS OF FORTRAN

1. Introduction

The British Computer Society's Specialist Group on Fortran held its first meeting on Tuesday 6th January 1970. As a step towards standardizing and developing the language it was decided that a report should be produced to "review implementations against the existing ASI standards(1) in regard to syntax and semantics". This is that report.

1.1 The Aims of this Report

This report examines some widely used compilers and shows where they diverge from the ASI standards. It also examines some extra facilities which are not in the ASI standards but are none the less widely implemented or popular at individual installations.

The only compilers examined in this report are those with which the authors have had experience. They felt it foolish to include any more because only practical experience can show up the subtle shades of interpretation given to the ASI standard by different writers of compilers. Nevertheless the range of compilers dealt with here does cover those used by a large section of the Fortran programming community in Britain.

This report deals only with Fortran compilers; not Basic Fortran.

1.2 The Form of the Report

Identifying the differences between different dialects of Fortran is a Herculean task which has been valiantly undertaken by others (2) (3) (4) (5) & (6). In this report a feature is included only if:

(a) it deviates from the ASI standards on at least one compiler, or
(b) there are compelling extensions to it.

Features which have been implemented in strict accordance with the standards are not included.

For every facility included here a reference is made to the relevant paragraphs of the ASI Standard (1). For example the section on the CALL statement is introduced by:

5.4 CALL ASI 7.1.2.4 & 8.4.2

This approach has been adopted in preference to reproducing the formal descriptions, thereby making this report so long that it would never be read.

To simplify the task of keeping this report up-to-date each facility is presented on its own sheet. These are grouped into chapters and arranged alphabetically within each. Chapters cover general topics such as the flow of control within a subprogram or input and output.

The notation used is that of the ASI Standard (1).

1.3 The Compilers Examined

These are the compilers reviewed. Each is given a letter which becomes its name for the rest of the report. The references are to the manufacturer's manuals:

Compiler A is the Fortran Compiler on Control Data Corporation's 6600 computer (8).

Compiler B is the L-level Compiler on Honeywell's 200 computer(9).

Compiler C is the G-level compiler on International Business Machines' System 360 computer(10).

Compiler D is the 32K Disc Compiler called XFAT/2C on International Computers' 1900 computer(11).

Compiler E is the Fortran IV Compiler on International Computers' System 4 computer(12).

Compiler F is the Fortran V Compiler on Univac's 1108 computer(13).

1.4 Acknowledgements

The authors thank the British Computer Society's Fortran Specialist Group for this opportunity to make constructive comments on a problem which has exasperated both of them on many occasions. A list of people who have contributed to their knowledge of the quirks of each compiler is too long to include, and it would in any case be unfair to lay the blame for any inaccuracies this report may contain at anyone's feet but the authors'.

1.5 References

1. 'American Standard Fortran', published by the American Standards Association Incorporated as Standard X3.9-1966.

2. 'A Study of Fortran Compatibility' by C.F. Schofield published by the University of London Atlas Computing Service in 1968.

3. 'Standard Fortran Specification' published by Mott, Hay and Anderson in 1967.

4. 'Standard Fortran Programming Manual' published by the National Computing Centre.

5. 'Fortran Programming' by Fredric Stuart published by John Wiley & Sons, Inc. in 1969.

6. 'Programming Languages' by Jean E. Sammet published by Prentice-Hall, Inc. in 1969.

7. 'Fortran - A Comparative Study' by P. Bryant published by the Science Research Council, Atlas Computer Laboratory in 1968.

8. '6400/6500/6600 Computer Systems. Fortran Reference Manual' published by the Control Data Corporation in 1969, reference '60174900 Rev.C'.

9. 'Series 200/Mod 2 (Extended) Fortran Compiler L' published by Honeywell in December 1968, reference 'File Number 123.1305.002L.3-718'.

10. 'System/360. Fortran IV Language' published by International Business Machines in 1966, reference 'Form C28-6515-6'.

11. 'Fortran. 1900 Series' published by International Computers Limited in 1968, reference 'Technical Publication 4088(TL1167)'.

12. 'System 4. Fortran Reference Manual' published by International Computers Limited, reference 'Technical Publication 1060'.

13. '1108 Fortran V Programmers Reference' published by Univac, reference 'UP4060 Rev.1.'.

CHAPTER 2

THE WRITTEN FORM OF FORTRAN

This chapter examines the way Fortran program is prepared for compilation. Statement-labels are not discussed because each compiler appears to provide these in accordance with the Standard (ASI 3.4).

2.1 CHARACTER SET ASI 3.1

EXTENSIONS: Compilers C, D, E and F include the characters ' and &. Compiler F also includes the character @ and Compiler B the character : .

2.2 COMMENTS ASI 3.2.1

DIFFERENCES: Compiler C deletes the 31st and subsequent comment lines if there are more than 30 contiguous comment lines between two consecutive statements.

EXTENSIONS: 1. Continuations

Compilers D and F accept comments immediately before a continuation line.

2. Page Control

Compiler A accepts $ or * in place of C to introduce a comment. It also accepts a decimal point and this causes a new page to be started at compilation time.

2.3 NAMES ASI 3.5

EXTENSIONS:

1. Length of Names

Compiler D permits up to 32 characters in a name, Compiler A up to 7 (but a name may not be the letter 0 followed by 6 digits) and although Compiler C accepts long names it treats only the first six characters as significant and prints a warning if more than six appear.

2. Additional Characters

Compilers C and E treat the character $ as alphabetic.

2.4 ORDER OF STATEMENTS ASI 9.1.1, 9.1.2, 9.1.3 & 9.1.5

REMARK: The Standard (in effect) states that the order of statements for a main program, subroutine or function must be:

(a) SUBROUTINE or FUNCTION statement, if appropriate.
(b) Specification statements, if any.
(c) Statement function definitions, if any.
(d) Executable statements; at least one must be present.
(e) END

FORMAT statements may be interspersed with (b), (c) and (d) and DATA statements with (c) and (d).

The order of statements in a BLOCK DATA subprogram is discussed under the heading 'BLOCK DATA'.

DIFFERENCES: 1. Heading of Main Program

Compiler A requires and Compiler E permits a PROGRAM statement at the start of the main program. Compiler D requires a MASTER statement.

2. Constraints on the Order

Compiler D requires statements to be ordered more rigidly than the Standard. Compilers A, B and E do not specify an order although some notes on ordering do appear in their manuals against individual statements such as COMMON and Statement Functions. Compiler F recommends an order but states that deviations from it are not necessarily wrong.

EXTENSIONS: Compiler C requires that IMPLICIT declarations appear between (a) and (b), that NAMELIST statements obey the same rules as FORMAT statements and that DATA statements may be interspersed with (b) as long as initialized variables do not appear in later specification statements.

2.5 WRITING A STATEMENT ASI 3.2.3, 3.2.4 & 3.3

DIFFERENCES: Compiler A requires that columns 1 to 5 of a continuation line are blank.

EXTENSIONS: 1. Comments

Compiler F accepts the character @ in a statement and ignores the remainder of the line, thus allowing comments on any line. The character @ in a Hollerith constant is treated as part of the constant.

2. Multiple Statements

Compiler A accepts several statements on one line, each separated by the character $. FORMAT statements and DATA statements may not use this facility.

CHAPTER 3

DECLARING AND MANIPULATING DATA

This chapter examines the way data are declared, allocated storage-space and manipulated. The way an array is mapped onto the computer's store is not described because each compiler appears to do this in accordance with the Standard (ASI 7.2.1.1.1). For the same reason arrays and array-elements are not discussed (ASI 5.1.3 and 5.1.3.1).

3.1 ARITHMETIC EXPRESSIONS ASI 6.1 & 6.4

EXTENSIONS:

1. Mixed-Mode

All six compilers permit mixed-mode expressions except Compilers D and F which do not allow Complex and Double to be mixed. Compiler A permits arithmetic operations on logical variables which are treated in this context as integers.

2. Unary Operators

Compiler F permits a unary minus to follow another operator in some situations, for example A/B. Compiler A permits unary minus to follow two stars.

3. Exponentiation

Compilers C and E permit A**B**C and evaluate it from right to left.

Compilers B, C, D and F permit a Real or a Double to exponentiate an Integer. Compiler F permits a Real to exponentiate a Complex and Compiler A permits an Integer to exponentiate a Logical.

3.2 ASSIGNMENT ASI 7.1.1.1 & 7.1.1.2

EXTENSIONS:

1. Mixed Assignment

Compilers A, B, C, E and F permit these mixed assignments in addition to the Standard's:

I=C, R=C, D=C, C=I, C=R, C=D

Compiler F also permits:

I=typeless, L=typeless, R=typeless, I='ch.string'

Compiler A permits any variable except a logical to be assigned a masking expression, and also allows any other type of expression to be assigned to a logical. Compiler F permits assignment into any binary subfield of a variable.

2. Multiple Assignment

Compilers A and D permit multiple assignment.

3.3 COMMON ASI 7.2.1.3

DIFFERENCES:

1. Alignment of Variables

Because of the architecture of the computers for Compilers C and E COMMON storage can not in general be mapped into the store in accordance with the Standard (ASI 7.2.1.3.1.1). Compiler E requires the programmer to arrange Common variables in decreasing order of length (i.e. 16-byte variables first, then 8-byte variables, then 4, then 2 and then 1). Compiler C recommends this procedure; if it is not followed the program will run, but will 'lose considerable object-time efficiency'.

2. Limits

Compiler A limits the number of labelled Common Blocks to 61. Compiler B permits only 16 in each chain (a program can be as many as 999 chains).

EXTENSIONS:

1. DATA and COMMON

Compilers A and F allow initialization of variables in a Common Block in any subprogram in which the Block appears.

2. Name of Block

Compiler A permits the name of a Common Block to be up to seven characters. A nonnegative integer may also be used to identify a Common Block.

3. Lengths of Blocks

Compilers D and F do not required a labelled Common Block to be the same length everywhere it is declared. In practice this is also true of Compiler C provided that labelled Common Blocks which appear in BLOCK DATA are not declared with greater length in another subprogram. Compiler A permits variation but requires the longest to be loaded first.

3.4 DATA ASI 7.2.2

DIFFERENCES: Compiler C requires the types of a variable and its initializing constant to agree. Compiler E permits the types to vary and converts the constant in a suitable way.

EXTENSIONS:

1. Length of Lists

Compiler A permits the length of a list of variables to differ from that of its list of initializing constants, in which case the trailing part of whichever list is longer is ignored.

Compilers C and E permit a list of constants to be shorter than its list of variables if the last variable is an array. Compiler D also permits variables to be names of arrays.

Compiler F has an option to work either as Compiler C or to permit the list of constants to be shorter than the list of variables.

2. Form of Lists

Compilers A, B and F permit the list of variables to include arrays and implied DO-loops as permitted in I/O lists.

Compiler A accepts an older syntactic form of DATA statement.

3. Constants

Compilers C and E permit the characters T and F to be written in place of the full form of the logical constants.

Compilers A, D and F permit a Hollerith constant to spread across more than one variable.

Compilers A and F allow octal constants and Compiler C allows hexadecimal constants.

3.5 DECLARING AN ARRAY ASI 7.2.1.1

DIFFERENCES: Compiler C requires integer variables used as declarator-subscripts to be four bytes long.

EXTENSIONS:

1. Number of Dimensions

Compilers C, E and F permit up to 7 dimensions and Compiler D up to 32.

2. Variable Dimensions

Compilers C, D, E and F permit integer variables used as declarator-subscripts to be in Common.

3. Size Limits

Compiler A can not process an array with more than 131071 elements.

3.6 DIMENSION ASI 7.2.1.2

EXTENSIONS: Compiler F enables variables to be given initial values as well as being defined in a DIMENSION statement.

3.7 EQUIVALENCE ASI 7.2.1.4

DIFFERENCES:

1. Misalignment

Compiler E's manual does not mention dangerous possibilities of misaligning array-boundaries. Compiler C does not require proper alignment but much efficiency is lost if variables are not properly aligned.

2. Restriction on use

Compiler F does not allow equivalenced variables to be used as subscripts or DO-loop parameters.

EXTENSIONS:

1. Arrays

Compilers A, D and F permit an array's name to be written in place of its first element.

2. Arguments

Compiler F permits a formal argument of a subprogram to appear in an EQUIVALENCE statement and this forces it to be called by name. (A similar effect can be achieved with Compiler C by using two obliques. Compiler F ignores two obliques).

3.8 EXTERNAL ASI 7.2.1.5

EXTENSIONS: Compiler F permits an EXTERNAL statement to be omitted if the name of a subprogram which should appear in an EXTERNAL statement is referred to directly.

Compiler A allows the Fortran II form of the external statement, which has F in column 1.

3.9 THE INITIAL VALUES OF VARIABLES ASI 10.2.3

REMARK: The Standard, and all manuals, state that all values not initialized by DATA or similar statements are undefined when a program begins. Compiler F places zeros in each variable and this fact is sometimes assumed by programmers. Compilers A, C, D and E start with garbage in each variable and what Compiler B does is not known.

3.10 PRECISION ASI 5.1.1 & 7.2.3.1.1

REMARKS: The Standard does not define the precisions of each type of data except to require that double precision be more precise than single (ASI 4.2.3) and that each component of a complex variable should be as precise as a real (ASI 4.2.4).

DIFFERENCES: These are the precisions of each compiler when using the Standard's storage-units for each variable:

COMPILER	A	B	C	D	E	F
LARGEST INTEGER+1	2^{59}	2^{23}	2^{31}	2^{23}	2^{31}	2^{35}
LARGEST REAL	10^{322}	10^{616}	10^{75}	10^{76}	10^{75}	10^{38}
SMALLEST +ve REAL	10^{-294}	10^{-616}	10^{-78}	10^{-76}	10^{-78}	10^{-38}
DECIMAL PRECISION	14.4	10.8	7.2	11	7.2	8.1
LARGEST DOUBLE	10^{322}	10^{616}	10^{75}	10^{76}	10^{75}	10^{308}
SMALLEST +ve DOUBLE	10^{-294}	10^{-605}	10^{-78}	10^{-76}	10^{-78}	10^{-308}
DECIMAL DOUBLE P.	28.9	21.7	16.8	20	16.8	18.1
MOST CH.S IN TEXT	∞	∞	255	∞	255	∞

EXTENSIONS:

1. Large Constants

Compilers C and E assume that a constant with more than seven digits is double precision.

2. Logical Constants

Compiler A allows logical constants to be written as .T. and .F.

3. Octal and Hexadecimal Constants

Compilers A, B and F accept octal constants and compiler C accepts hexadecimal constants.

3.11 RELATIONAL AND LOGICAL EXPRESSIONS ASI 6.2, 6.3 & 6.4

EXTENSIONS:

1. Mixed-Mode

Compilers A, B, C, D, E and F permit arithmetic expressions of mixed modes to be related. Complex expressions may not be related with Compilers C and E but Compilers B and F permit complex expressions to be related by EQ. and NE. Compiler A allows complex expressions but considers only their real parts.

3.12 STATEMENT-FUNCTIONS ASI 8.1

DIFFERENCES: Compiler A limits the number of arguments to 60. Compiler F requires all the arguments to be used in the expression.

EXTENSIONS: Compiler F allows a generalised statement function (DEFINE statement); this permits a statement function to have no arguments.

3.13 SUBSCRIPTS ASI 5.1.3.2 & 5.1.3.3

REMARK: Compilers B and E are remarkable in that they limit the complexity of a subscript-expression to that defined in the Standard.

DIFFERENCES: The manual for Compiler F is unusual in stating that it is the programmer's responsibility to ensure that the value of a subscript-expression remains within its declared limits. Although not stated this is also true of the other compilers except when using Compilers C and D in 'debug' mode.

EXTENSIONS: Compiler D permits a subscript to be any integer expression and Compiler C accepts any integer or real expression. Compiler A accepts an expression composed of integer constants and unsubscripted variables compounded by the operators +, -, * and /. It does not allow this more general form in an I/O list. Compiler F accepts expressions of the form:

$\alpha_1 M_1 \alpha_2 M_2 \ldots \alpha_v M_v$

where α_L is + or - (α_1 may be null)

and M_L is $K_1 * K_2 * \ldots K_\tau$

where K_L is an integer constant, unsubscripted variable or parameter and no more than one K in each M may be the index of a DO-loop.

Compiler A permits reference to an element of an array with fewer subscripts than the number declared for the array. Subscripts may be omitted from the right. Any so omitted are assumed to have value unity.

3.14 TYPES OF DATA ASI 4.2 & 7.2.1.3.1.1

REMARK: The Standard defines these types: Integer, Real, Double Precision, Complex, Logical and Hollerith. Integer, Real and Logical variables occupy one 'storage unit' and Double Precision and Complex two.

DIFFERENCES:

1. Terminology

Compiler D consistently refers to Hollerith data as 'Text'.

2. Lengths

The reader is referred to the section on Common for situations in which Compilers C and E violate the Standard's requirements concerning the length of each variable.

EXTENSIONS:

1. Lengths

Compilers C, D and E permit integers to occupy half a storage unit. Logicals may be one quarter of a unit with Compilers C and E and one half a unit with Compiler D. Compilers C and E permit complex to occupy four units and Compiler D can squeeze double precision into one unit.

2. Strings of Binary Digits

Compiler F permits 'typeless' variables in which bit-strings may be manipulated and Compiler A provides facilities for processing bit-strings in logical variables.

3.15 TYPE STATEMENTS ASI 7.2.1.6

EXTENSIONS:

1. Lengths

Compilers C and E permit the length of each variable in bytes to be specified. Compiler F accepts these extended syntactic forms and ignores them except for treating REAL*8 as Double Precision. Compilers C and E also permit the lengths to appear after each variable.

2. Initialization

Compilers C, E and F provide forms which enable data to be initialized as it is typed although Compilers C and E do not allow this facility to be used in the DOUBLE PRECISION statement.

3. Other Syntactic Variations

Compiler A accepts 'DOUBLE' in place of 'DOUBLE PRECISION' and permits 'TYPE' to precede any type-statement.

CHAPTER 4

FLOW OF CONTROL WITHIN A SUBPROGRAM

This chapter reviews features of Fortran which affect the flow of control within a subprogram, including the assignment of values to controlling variables. The unconditional GO TO statement (ASI 7.1.2.1.1) is not discussed because this appears to be uniformly implemented by all of the compilers.

4.1 ARITHMETIC IF ASI 7.1.2.2

EXTENSIONS:

1. Type of expression

Compiler F allows the expression to be 'typeless'. Compiler A allows the expression to be complex, in which case only the real part is considered.

2. Labels

The label fields may be left blank (Compiler F) or be zero (Compiler D) to indicate the next statement. The may be replaced by ASSIGNed variables on Compiler F.

4.2 ASSIGN ASI 7.1.1.3

DIFFERENCES: Compilers C and E require the variable being assigned a label to be four bytes long.

EXTENSIONS: Compiler F permits the variable to be any type as long as it occupies one word.

4.3 ASSIGNED GO TO ASI 7.1.2.1.2

REMARKS: The Standard requires that the variable is one of the labels in the list.

DIFFERENCES: 1. Type of Variable

Compilers C and D insist that the variable is not a short integer.

2. Checking

Compilers A, C, D and F (and possibly others) do not check at execution time that the variable corresponds to a label in the list. However Compilers C and F (and possibly others) do check at compilation time that the labels in the list are legitimate.

EXTENSIONS: 1. Syntax

The comma and the list of labels may be omitted for Compilers A, D and F.

2. Type of Variable

Compiler F permits the variable to be any type as long as it occupies one word.

4.4 COMPUTED GO TO ASI 7.1.2.1.3

EXTENSIONS:

1. Syntax

The comma between the label list and the variable is optional for Compiler A.

2. Label List

The labels may be replaced by zero indicating the next statement, on Compiler D and by ASSIGNed variables on Compiler P.

3. Error Action

If the variable is out of range the actions taken are:

Compilers A and F: program terminates in error mode.

Compilers C, D and E: program continues with next statement.

Compiler B: action is not defined.

4.5 CONTINUE ASI 7.1.2.6

DIFFERENCES: Compiler A insists that a CONTINUE statement is labelled.

4.6 DO ASI 7.1.2.8

DIFFERENCES:

1. Depth of Looping

Compilers C and F both permit up to 25 loops to be nested.

2. Extended Range

Compiler A requires that if a transfer is made from a loop's extended range back into the loop then the transfer may not take place to the terminal statement of the loop unless its label has already appeared in a GO TO or ASSIGN statement.

EXTENSIONS:

1. Terminal Statements

Compiler F permits the terminal statement of a loop to be non-executable. Compiler E permits a GO TO as a terminal statement as long as it is preceded by a logical IF clause.

2. Controlling Parameters

Compilers A and F permit the initial and the terminal parameters to he zero or negative. Compiler F also permits the initial parameter to be greater than the terminal parameter and the incrementing parameter to be negative.

Compiler A permits a controlling parameter to be changed within a loop.

Compiler D accepts any integer expression in place of a controlling parameter.

3. Extended Range

Compiler D has less restrictive rules than those in the Standard for transferring in and out of DO-loops.

4.7 LOGICAL IF ASI 7.1.2.3

EXTENSIONS: Compiler A allows the trailing statement to be replaced by two statement-labels to which control passes depending whether the logical is true or false.

4.8 PAUSE ASI 7.1.2.7.2

DIFFERENCES: Compilers C and E treat the digits as decimal not octal.

EXTENSIONS: Compilers B, C, D, E and F permit strings of characters in place of the digits. Compilers C and E allow up to 255 characters and they must be enclosed in quotes. Compiler B allows up to 40 characters enclosed in colons and Compiler D up to 40 characters enclosed in quotes. Compiler D also allows up to 5 alphanumeric characters not surrounded by other symbols and Compiler F allows up to 6 such characters.

4.9 STOP ASI 7.1.2.7.1

DIFFERENCES: Compilers C and E treat the digits as decimal not octal.

EXTENSIONS: Compiler B allows a string of up to 40 characters enclosed in a pair of colons to replace the digits and Compiler D allows up to 40 characters enclosed in quotes. Compiler D also allows up to 5 characters not enclosed by other symbols and Compiler F allows 6 such characters.

CHAPTER 5

CONNECTIONS BETWEEN SUBPROGRAMS

This chapter describes those features of Fortran concerned with subprograms and their relationships. The circumstances in which each compiler calls by value or by name are not discussed because the picture presented by the manuals is a blurred one. The point is discussed in the Standard in Sections 8.3.2 and 8.4.2 and the interested reader is referred to Reference 7 given in Section 1.5 of this report.

5.1 ARGUMENTS OF FUNCTIONS ASI 8.3.2

DIFFERENCES:

1. Limits

Compiler A limits the number of arguments to 60 and Compiler F to 64.

2. Arguments

Compiler A debars the names of I/O buffers from an argument-list and Compiler C forbids ASSIGNed variables and hexadecimal constants.

3. Dimensionality

Compiler B requires an array to have the same dimensionality in called and calling subprograms.

EXTENSIONS:

1. Hollerith Constants

Compilers A, C, D and F allow Hollerith constants as arguments.

2. Labels

Compiler E allows the label of a FORMAT statement preceded by & as an argument. Compilers D & F allow the label of an executable statement as an argument; the label is preceded by & for Compiler D and by & or $ for Compiler F.

3. Names

Compilers D and E allow a NAMELIST name as an argument and Compiler F allows the name of an internal subprogram.

5.2 BASIC EXTERNAL FUNCTIONS ASI 8.3.3

DIFFERENCES: DMOD is intrinsic with Compilers C, E and F.

EXTENSIONS:

1. Compiler A

DABS, SNGL, DBLE, ACOS, ASIN, DSIGN, IDINT, LEGVAR, LENGTH, RANF, TAN, LOCF, XLOCF.

2. Compiler C

CDEXP, CDLOG, ARSIN, DARSIN,
ARCOS, DARCOS, CDCOS, CDSIN, TAN, DTAN,
COTAN, DCOTAN, CDSQRT, DTANH, SINH, DSINH,
COSH, DCOSH, CDABS, ERF, DERF, ERFC, DERFC,
GAMMA, DGAMMA, ALGAMA, DLGAMA.

3. Compiler D

EXP10, ALOG2, TAN, COT, SINH, COSH, COTH,
ASIN, ACOS, ACOT, ASINH, ACOSH, ATANH, ACOTH.

4. Compiler E
CDEXP, CDLOG, CDSQRT, DTANH, CDABS, CDSIN, CDCOS.

5. Compiler F
TAN, DTAN, CTAN, ASIN, DASIN, ACOS,
DCOS, SINH, CSINH, DSINH, COSH, DCOSH,
CCOSH, DTANH, CTANH, CBRT, DCBRT, CCBRT.

5.3 BLOCK DATA ASI 8.5 & 9.1.4

DIFFERENCES:

1. Order of Statements

Compiler D restricts the order of statements more rigidly than the Standard.

2. Many BLOCK DATA Subprograms

If, using Compiler C, a block is initialized in more than one BLOCK DATA subprogram then all but one of the subprograms are ignored.

Compiler D does not permit a block to be initialized by more than one BLOCK DATA subprogram.

EXTENSIONS:

1. Order of Statements

Compilers C and F permit specifications followed by DATA statements followed by more specifications followed by more DATA statements and so on in accordance with the rules of ASI 9.1.4.

2. Many BLOCK DATA Subprograms

Compiler A permits a BLOCK DATA Subprogram to he named and this makes for precise control of segmentation.

5.4 CALL ASI 7.1.2.4 & 8.4.2

REMARKS: Although all of the compilers accept the same form of CALL statement they vary from the standard in the arguments they permit. The standard requires that all actual arguments agree with dummy arguments in number and in type but none of the systems is known to check this.

DIFFERENCES:

1. Number of Arguments

Compilers A and F publish a limit on the total number of arguments (60 and 64 respectively).

2. Dimensionality

Compiler B requires an array to have the same dimensionality in a called subprogram as in its calling subprograms. This is not a requirement of the standard (ASI 7.2.1.1.1).

3. Buffers

Using an array as an I/O buffer on Compiler A precludes its use as an argument.

EXTENSIONS:

1. Labels

Compilers C, D, E and F permit statement-labels as arguments if preceded by the character &. (Compiler F permits $ in place of & and so does Compiler C if the program is in BCD). Compiler A permits a statement-label as an argument when calling two particular system-routines only; the label is followed by the letter S.

2. Internal Procedure

Compiler F permits the name of an internal procedure as an argument.

3. Formats and Namelists

Compiler E permits a format-number as an argument (preceded, like a label, with &). Compilers D and E permit the transmission of a NAMELIST name.

5.5 FUNCTION ASI 8.3.1

DIFFERENCES: Compiler F does not permit a dummy argument corresponding to a Hollerith constant in the invoking reference to be a Double Precision or Complex array.

EXTENSIONS:

1. Length of Result

Compilers C and E enable the length of the function to be controlled by writing *b after the name, where b is the required number of bytes. Compiler F accepts and ignores such length-controls unless the function is Real and *8 appears in which case it treats it as Double Precision.

2. Type of Result

Compilers C, E and F allow a function to be typed by a type statement in the body of the function.

3. Arguments

Compilers D and F accept * as a dummy argument corresponding to a statement number in the function's invokation. (Compiler F also accepts in place of *). Compiler D accepts a dummy argument enclosed in obliques and this causes reference by location. Compiler F ignores obliques around a dummy argument. Including a dummy argument in an EQUIVALENCE statement causes it to be taken as a call by name.

4. Form of Statement

Compiler A allows the form FORTRAN n t FUNCTION where n is II, IV or VI and t is the type.

5.6 INTRINSIC FUNCTIONS ASI 8.2

DIFFERENCES: Compiler A treats these functions as external: DABS, SNGL, DBLE, DSIGN, IDINT. Compilers C and E treat Maximum and Minimum Functions as external although the H-level version of Compiler C treats them as intrinsic.

EXTENSIONS:

1. Compiler A

AND, COMPL, OR.

2. Compiler B

DFLOAT, IAND, IOR, ICOMPL, IEXCLR.
The manual does not distinguish between intrinsic and external functions.

3. Compiler C

DMOD, DFLOAT, HFIX, DCONJG.

4. Compiler D

NINT, ANINT.

5. Compiler E

DMOD, DINT, HFIX, DFLOAT, CSNGL, DREAL, DIMAG, CDBLE, DCMPLX, DCONJG.

6. Compiler F

AND, OR, XOR, BOOL, COMPL, LOC, FLD, DINT, DMOD, DDIM.

5.7 OWN VARIABLES IN SUBPROGRAMS ASI 10.2.6

REMARKS: All six compilers preserve values of local variables and arrays in subprograms after control has returned to the calling program. On subsequent entries to the subprogram all local variables have the values they had at the last execution of a RETURN in that subprogram, unless that program has been overlaid between subprogram references. In this case variables in DATA and similar statements are reinitialized and other variables are undefined, except by Compiler D which preserves other variables from the previous entry. (This decision is under review by ICL).

The Standard states that uninitialized local variables are undefined on re-entry to a subprogram. Many programs rely upon the preservation of variables described above.

5.8 RETURN ASI 7.1.2.5

EXTENSIONS:

1. In Main Program

Compilers A, C, E and F allow RETURN in a main program and treat it as STOP.

2. Variable Return

Compilers C, D, E and F allow RETURN i which causes return to a statement-number passed as an argument to the subprogram. All four compilers allow the statement in a subroutine; only Compilers D and F allow it in a function. Compilers C and E require that if i is an integer variable it must not be a short integer. Compiler F counts arguments differently from the others. In Compiler F execution of this statement when the i'th argument is not a statement-number causes an error; RETURN 0 is the standard error-exit from a program.

5.9 SUBROUTINE ASI 8.4

DIFFERENCES: Compiler F does not permit a dummy argument corresponding to a Hollerith constant in the invoking reference to be a Double Precision or Complex array.

EXTENSIONS: Compilers C, D and E permit a dummy argument to be a star and this corresponds with & in the calling statement. They also accept a dummy argument enclosed in obliques and this causes reference by location. Compiler F permits a dummy argument to be a $ or * and this corresponds to a statement-number in the calling statement. It ignores obliques around dummy arguments but the appearance of a dummy argument in an EQUIVALENCE statement causes it to be taken as a call by name.

CHAPTER 6

INPUT AND OUTPUT

This chapter discusses the facilities in Fortran for manipulating a computer's peripheral devices.

6.1 BACKSPACE ASI 7.1.3.3.2

DIFFERENCES: Compiler C requires that the variable containing the unit-number is four bytes long.

6.2 ENDFILE ASI 7.1.3.3.3

DIFFERENCES: Compiler C requires that the variable containing the unit-number is four bytes long.

6.3 FORMAT ASI 7.2.3 & 7.1.3.4

REMARKS: The Standard requires that the number of characters produced by an output-conversion must not exceed the width of the field. It does not specify what should happen if a number is too large for a field.

DIFFERENCES:

1. Vertical Spacing

Compiler B does not allow the symbol + as a control-character to prevent the printer advancing.

2. Printing Zero

Compiler C always prints '0.0' for exact real zero irrespective of the FORMAT.

3. Printing Complex Numbers

The manual for Compiler F requires that the two components of a complex number are printed on the same line, but this does not appear to be a necessary constraint in practice.

4. Limits

Compiler E restricts any number used with a FORMAT (except a literal) to less than 256.

EXTENSIONS:

1. Free Format

Compilers D, E and F provide format-free input, Compilers D and E by accepting zero values for field-widths, Compiler E by accepting a special Code Y and Compiler F by accepting an empty format.

2. Insufficient Field Width

If a number overflows its field Compilers C, E and F print stars in its place; Compiler A prints a single star at the beginning of the field and then prints as much of the number as fits in the field. Compiler D prints a star and the number in full and ignores the field-width.

3. Additional Codes

Compilers B, C, D, E and F permit G-Format to handle Integers, Logicals and Doubles.

Octal I/O (using O-Format) is available with Compilers A, B and F. Compilers C and E provide Z-Format for handling hexadecimal I/O.
Controlled 'tab' settings (T-Format) are available with Compilers B, C, D, E and F.

Right-justified character I/O (R-Format) is available with Compilers A and F.

Text may appear in quotes with Compilers C, D, E and F and surrounded by a pair of stars with Compiler A.

4. Vertical Spacing

Additional carriage-control characters are provided by Compilers B, C, D and E. Compilers A and F treat any carriage-control character not allowed by the Standard as blank.

5. Interaction with I/O List

In some circumstances Compiler B accepts a variable in an I/O List which does not match the type of the conversion code and does a suitable conversion.
Compiler A has two rules for the interaction between an I/O List and its FORMAT; one ASI and one its own. A switch can be set to control which rule is used.

6. Limits

Compiler B restricts all field-widths to less than 64 characters.

7. Reference to Format

Compiler A permits the reference to a Format to be a simple variable or a subscripted variable. Compiler F accepts a simple variable. Compilers C and F permit text in the form nH in a run time Format. Compiler C also allows text in quotes for output only.

8. Repeated Input

Compilers D and F provide facilities for repeatedly rereading the same input.

6.4 I/O LISTS ASI 7.1.3.2.1

DIFFERENCES: Compiler B restricts the depth of implied loops to 3.

EXTENSIONS: Each compiler allows a subscript in an I/O List to take the forms it permits elsewhere (these are described in Section 3.13) except Compiler A which restricts the complexity of a subscript in an I/O List to the strict form laid down by the Standard (ASI 5.1.3.3).

6.5 READ AND WRITE ASI 7.1.3

EXTENSIONS:

1. Checking Errors and End-of-File

Compilers B, C, D, E and F include optional clauses of the form 'END=label' and 'ERR=label' in the READ statement for checking if the end of a sequential file is met or if errors have occurred in the transfer. Compiler F provides the, same facilities with WRITE. Compiler A enables IF statements to check if these conditions have occurred.

2. Name List

Compilers A, C, E and F permit a reference to a Format to be a reference to a Name List. Compiler D permits reference to a Name List in a WRITE statement only. Compiler F accepts the 'END=' and 'ERR=' in conjunction with a Name List.

3. Free Format

Compiler B permits LIST in place of the Format and this provides I/O free of format restrictions.

6.6 REWIND ASI 7.1.3.3.1

DIFFFRENCES:

1. Lengths

Compiler C requires the variable to be four bytes long.

2. Tape Marks

Compilers B and D mark the tape before rewinding.

3. Multi-file Magnetic Tapes

If the unit refers to a magnetic tape file, Compilers A, B and F assume that REWIND refers to the tape reel and Compilers C and E that it refers to the current file. (Compiler D does not permit multi-file tapes).

6.7 UNIT NUMBERS ASI B6

REMARKS: Most Compilers provide default numbers for referring to the standard input device, the standard output device and so on. These numbers can depend not only on the compiler but also on the installation. Most installations limit the possible range of unit-numbers. The default numbers may be overridden in these ways:

(a) Dynamically, by calling subroutines, with Compiler B.
(b) Statically by specifying the unit-numbers in the Job Control Language, with Compilers C, D and E.
(c) Statically, by specifying the unit-numbers in the PROGRAM Card, with Compiler A.
(d) Statically, by supplying a program containing a table of unit-numbers to override the standard table in the Fortran Library, with Compiler F.

CHAPTER 7

SOME POPULAR EXTENSIONS

This chapter describes some well-tried extensions to Fortran which involve either new statements or clothing old statements with entirely new meanings.

To give further indication of a facility's popularity Fortran Compilers provided by other manufacturers are included in this chapter where information was available to the authors. Their names are:

G Atlas Fortran V
H Burroughs
I DEC
J GE
K Hughes
L Raytheon
M RCA
N SDS

7.1 ABNORMAL

REMARKS: Compilers E and F provide the ABNORMAL statement for classifying Functions so that the compilers can generate more efficient code than they could without this 'hint'.

AVAILABILITY: Compilers E, F.
Compiler N.

7.2 BUFFER IN AND OUT

REMARKS: Compiler A enables the programmer to control the way the computer simultaneously deals with his input, output and computation. Similar facilities are available by Library Subprograms with Compiler F.

AVAILABILITY: Compiler A.

7.3 DEBUGGING AIDS

REMARKS: Compiler B provide selected tracing of variables and labels.

Compiler C includes a DEBUG package for checking that the bounds of selected arrays are not violated and for tracing labels, variables and subprograms.

Compiler D provides three 'levels' of trace, Trace 0, Trace 1 and Trace 2. Trace 0 provides virtually no debugging aids. Trace 1 (which is the default state of the compiler) prints the results (if any) of the 100 statements carried out before an error caused the program to stop. Trace 2 checks accesses to all arrays for violation of their hounds.

Compiler F provides no debugging facilities but recommends the use of NAMELIST and its statements for controlling source-program, INCLUDE and DELETE.

AVAILABILITY: Compilers B, C and D.

7.4 DIRECT ACCESS I/O

REMARKS: Compilers A and F provide Library Subprograms for using I/O devices randomly.

Compilers C, D and E introduce the following new statements:

```
DEFINE FILE
READ (u'r,f,ERR=n) list
WRITE (u'r,f) list
FIND (u'r)
```

Compiler B provides the same facilities as C, D and E except that a colon replaces the prime and that 'f' can optionally be LIST.

AVAILABILITY: Compilers A, B, C, D, E and F

7.5 ENCODE AND DECODE

REMARKS: Compiler A enables data to be converted from binary form to BCD form and back again whilst still in core using ENCODE and DECODE.

Compiler F provides the same feature but the number of characters transformed is always 132 and this removes an argument from the statement.

Compilers D and E enable data to be 'read' from core instead of from a peripheral device. They do it in different ways.

AVAILABILITY: Compilers A, F.
Compiler N.

7.6 ENTRY

REMARKS: Compiler C's ENTRY statement is assumed to be standard for the remainder of this section. Compilers A, C, D, E and F implement ENTRY.

DIFFERENCES: Compiler F limits the number of ENTRYs in one subprogram to 50, and does not permit a reference to (in ENTRY name within a Function. (Compiler C equivalences such a name with the name of the Function).

Compilers A and F do not permit an ENTRY statement to be labelled.

Compiler E requires an ENTRY in a Function to be the same type as the function.

Compiler A does not include a list of arguments with an ENTRY statement. It assumes the same list of arguments as the subprogram in which it appears.

Compiler E does not require that a dummy argument listed at more than one entry point is consistently referenced by name or by value. If there are contradictions at different entries the first one is assumed.

AVAILABILITY: Compilers A, C, D, E, F.
Compilers H, J, K, L, M, N.

7.7 FORTRAN II I/O

REMARKS: Compilers A, C, E and F accept these Fortran II I/O statements:

```
READ f, list
PRINT f, list
PUNCH f, list
```

Compilers A and F accept these statements:

```
READ INPUT TAPE u, f, list
WRITE OUTPUT TAPE u, f, list
READ TAPE u, list
WRITE TAPE u, list
```

7.8 IMPLICIT

REMARKS: Compiler C's implementation of IMPLICIT is taken as standard for the remainder of this section. IMPLICTT is provided by Compiler B, C, E and F.

DIFFERENCES: Compilers B and F will not accept *b in the type.

According to its manual, Compiler F does not apply an IMPLICIT in a subprogram to its dummy arguments but this was not found to be the case.

EXTENSIONS: Compiler F allows an arbitrary number of IMPLICIT statements per subprogram. A type definition remains in effect until it is redefined by a following IMPLICIT statement.

Compilers B and F allow IMPLICIT to refer to Double Precision variables.

AVAILABILITY: Compilers B, C, E, F.
Compilers G, I, M, N.

7.9 NAMELIST

REMARKS: Compiler C's implementation of NAMELIST is taken as standard for the remainder of this section. NAMELIST is provided by Compilers A, C, D, E and F.

DIFFERENCES: Compiler F allows $ in data where Compiler C has &.

Compiler E does not allow END as a name and &END in data must be in a fixed position in the Record.

Compiler D does not allow NAMELIST input.

EXTENSIONS: Compiler A permits a dummy argument to appear in a NAMELIST's name.

AVAILABILITY: Compilers A, C, D, E, F.
Compilers H, I, J, L, M, N.

7.10 SERVICE ROUTINES

REMARKS: Compilers A, B, C, D, E and F provide Subroutines SLITE, SLITET, DVCHK and OVERFL.

Compilers A, B, C, E and F provide Subroutine EXIT.

Compilers A, B, C and E provide Subroutines DUMP and PDUMP.

Compilers A, B, D and F provide Subroutine SSWTCH.

Compiler D provides SWON and SWOFF.

CHAPTER 8

CONCLUSIONS

This chapter is a vehicle for the authors' prejudices. We attempt to draw some general conclusions from our review hoping that those who would attempt to extend Fortran can nimbly sidestep the deeper crevasses down which so many of their predecessors plummetted.

8. Conclusions

This review has shown that the range of divergences from and extensions to the Standard is wide, thus demonstrating that compiler-writers are an ingenious breed and also that the Standard is deliberately permissive in some areas (ASI B1.1). As machine-independence becomes more important these areas must be more tightly specified. Examples include the problems of "own" variables in subprograms, the precision of data and the interaction of a program with its environment.

Many compilers include "minor" extensions which their writers probably consider to be reasonable and within the spirit of the Standard. We would ask such writers to remember that a minor extension used many times throughout a program causes much more havoc than a major extension which occurs only sparsely.

Frequently extensions are made to a compiler which are not "general". For example one compiler permits subscripts to be general integer expressions except when used in an I/O list where they are restricted to the form permitted by the Standard. Some compilers provide sophisticated facilities for controlling the number of bytes each variable occupies but restrict variables used for certain activities (e.g. storing statement-labels or unit-numbers) to be four bytes long. We feel that a facility should not be added to Fortran unless it is general.

All compilers contain limitations affecting the size of programs they can process. No manufacturer known to the authors publishes all the limits built into his compilers. We feel that a manual is not complete without such a table of limitations. When this review mentions frequently the limitations of a particular compiler this is probably more a reflection of the manual's honesty than the compiler's limitations.