Annex B

(informative)

Decremental features

B.1 Deleted features

The deleted features are those features of Fortran 90 that were redundant and are considered largely unused. Section 1.7.1 describes the nature of the deleted features. The Fortran 90 features that are not contained in this standard are the following:

(1) Real and double precision DO variables.

The ability present in FORTRAN 77, and for consistency also in Fortran 90, for a DO variable to be of type real or double precision in addition to type integer, has been deleted.

(2) Branching to a END IF statement from outside its block.

In FORTRAN 77, and for consistency also in Fortran 90, it was possible to branch to an END IF statement from outside the IF construct; this has been deleted.

(3) PAUSE statement.

The PAUSE statement, present in FORTRAN 66, FORTRAN 77 and for consistency also in Fortran 90, has been deleted.

(4) ASSIGN and assigned GO TO statements and assigned format specifiers.

The ASSIGN statement and the related assigned GO TO statement, present in FORTRAN 66, FORTRAN 77 and for consistency also in Fortran 90, have been deleted. Further, the ability to use an assigned integer as a format, present in FORTRAN 77 and Fortran 90, has been deleted.

(5) H edit descriptor.

In FORTRAN 77, and for consistency also in Fortran 90, there was an alternative form of character string edit descriptor, which had been the only such form in FORTRAN 66; this has been deleted.

In this and other annexes, FORTRAN 66 is used as the informal name of the first international Fortran standard, ISO 1539:1972, which was technically identical to ANS X3.9-1966.

Recommendations are given in the following sections for those processors which extend the standard by implementing any of the deleted features.

B.1.1 Real and double precision DO variables

Replace rules R821 and R822 in section 8.1.4.1.1 by the following:

"R821 *loop-control* **is** [,] *do-variable* = *scalar-numeric-expr* , ■
■ *scalar-numeric-expr* [, *scalar-numeric-expr*]
or [,] WHILE (*scalar-logical-expr*)

R822 *do-variable* **is** *scalar-variable*

Constraint: The *do-variable* shall be a named *scalar-variable* of type integer, default real, or double precision real.

Constraint: Each *scalar-numeric-expr* in *loop-control* shall be of type integer, default real, or double precision real."

Replace the first part of section 8.1.4.4.1, up to and including the numbered item (1), by the following:

"When the DO statement is executed, the DO construct becomes active. If *loop-control* is

[,] *do-variable* = $scalar\text{-}numeric\text{-}expr_1$, $scalar\text{-}numeric\text{-}expr_2$ [, $scalar\text{-}numeric\text{-}expr_3$]

the following steps are performed in sequence:

(1) The initial parameter m_1, the terminal parameter m_2, and the incrementation parameter m_3 are of the same type and kind type parameter as the *do-variable*. Their values are established by evaluating $scalar\text{-}numeric\text{-}expr_1$, $scalar\text{-}numeric\text{-}expr_2$, and $scalar\text{-}numeric\text{-}expr_3$, respectively, including, if necessary, conversion to the type and kind type parameter of the *do-variable* according to the rules for numeric conversion (Table 7.10). If $scalar\text{-}numeric\text{-}expr_3$ does not appear, m_3 has the value 1. The value m_3 shall not be zero."

In section 8.1.4.4.1(3), replace "$(m_2 - m_1 + m_3) / m_3$" with "INT($(m_2 - m_1 + m_3) / m_3$)".

Replace rule R901 and the second constraint following it in section 9.4.2 by the following:

"R901 *io-implied-do-control* **is** *do-variable* = *scalar-numeric-expr* , ■
■ *scalar-numeric-expr* [, *scalar-numeric-expr*]

Constraint: The *do-variable* shall be a named scalar variable of type integer, default real, or double precision real.

Constraint: Each *scalar-numeric-expr* in an *io-implied-do-control* shall be of type integer, default real, or double precision real."

B.1.2 Branching to an END IF statement from outside its IF block

In section 8.1.2.2, second paragraph, change the second sentence to be, "It is permissible to branch to an END IF statement from within the IF construct, and also from outside the construct." In section 8.2, change the third paragraph to read, "It is permissible to branch to an END IF statement from within its IF construct, and also from outside the construct."

B.1.3 PAUSE statement

The definition of the statement is:

pause-stmt **is** PAUSE [*stop-code*]

Execution of a PAUSE statement causes a suspension of execution of the program. Execution shall be resumable. At the time of suspension of execution, the stop code, if any, is available in a processor-dependent manner. Leading zero digits in the stop code are not significant. Resumption of execution is not under control of the program. If execution is resumed, the execution sequence continues as though a CONTINUE statement were executed.

For completeness, "**or** *pause-stmt*" should be added to rule R216 in section 2.1.

Constraint: A pure subprogram shall not contain a *pause-stmt*.

B.1.4 ASSIGN, assigned GO TO, and assigned FORMAT

The definitions of the ASSIGN and assigned GO TO statements are:

assign-stmt **is** ASSIGN *label* TO *scalar-int-variable*

Constraint: The label shall be the statement label of a branch target statement or *format-stmt* that appears in the same scoping unit as the *assign-stmt*.

Constraint: *scalar-int-variable* shall be named and of type default integer.

assigned-goto-stmt **is** GO TO *scalar-int-variable* [[,] (*label-list*)]

Constraint: Each label in *label-list* shall be the statement label of a branch target statement that appears in the same scoping unit as the *assigned-goto-stmt*.

Constraint: *scalar-int-variable* shall be named and of type default integer.

Execution of an ASSIGN statement causes a statement label to be assigned to an integer variable. While defined with a statement label value, the integer variable may be referenced only in the context of an assigned GO TO statement or as a format specifier in an input/output statement. An integer variable defined with a statement label value may be redefined with a statement label value or an integer value.

When an input/output statement containing the integer variable as a format specifier (9.4.1.1) is executed, the integer variable shall be defined with the label of a FORMAT statement.

At the time of execution of an assigned GO TO statement, the integer variable shall be defined with the value of a statement label of a branch target statement that appears in the same scoping unit. Note that the variable may be defined with a statement label value only by an ASSIGN statement in the same scoping unit as the assigned GO TO statement.

The execution of the assigned GO TO statement causes a transfer of control so that the branch target statement identified by the statement label currently assigned to the integer variable is executed next.

If the parenthesized list is present, the statement label assigned to the integer variable shall be one of the statement labels in the list. A label may appear more than once in the label list of an assigned GO TO statement.

Further, "*assigned-goto-stm*t" should be added to the lists of prohibited statements in the first constraints to rules R829 and R833 in section 8.1.4.1.2. For completeness, "*assigned-stmt*" and "*assigned-goto-stmt*" should be added to rule R216 in section 2.1.

Add as a list item to the constraint about dummy arguments with INTENT(IN) attribute following R512 in section 5.1.2.3:

(11) In an *assign-stmt*.

In section 14.7.5, the following numbered item should be added: "Execution of an ASSIGN statement causes the variable in the statement to become defined with a statement label value.".

In section 14.7.5, the sentence in numbered item (10), second paragraph, "When a numeric storage unit becomes defined, all associated numeric storage units of the same type become defined" should have the following qualification added at the end, ", except that variables associated with the variable in an ASSIGN statement become undefined when the ASSIGN statement is executed".

In section 14.7.6, the following numbered item should be added: "Execution of an ASSIGN statement causes the variable in the statement to become undefined as an integer. Variables that are associated with the variable also become undefined."

In section 14.7.6, the following numbered item should be added: "A reference to a procedure causes part of a dummy argument to become undefined if the corresponding part of the actual argument is defined with a value that is a statement label value."

In section 12.6, add this item to the constraint that lists prohibited situations in pure subprograms:

(11) In an *assign-stmt*.

In section 9.4.1.1 add to rule R913: "***or*** *scalar-default-int-variable*" with the qualification that the *scalar-default-int-variable* shall have been assigned the statement label of a FORMAT statement that appears in the same scoping unit as the format.

B.1.5 H edit descriptor

In section 10.2.1, add the following line to rule R1016:

" **or** cH *rep-char* [*rep-char*] ..."

Add the following new rule with constraints, which logically follows rule R1016:

" *c* **is** *int-literal-constant*

Constraint: *c* shall be positive.

Constraint: *c* shall not have a kind parameter specified for it.

Constraint: The *rep-char* in the cH form shall be of default character type."

In the H edit descriptor, *c* specifies the number of characters following the H.

If a processor is capable of representing letters in both upper and lower case, the edit descriptors are without regard to case except for the characters following the H in the H edit descriptor and the characters in the character constants.

B.2 Obsolescent features

The obsolescent features are those features of Fortran 90 that were redundant and for which better methods were available in Fortran 90. Section 1.7.2 describes the nature of the obsolescent features. The obsolescent features in this standard are the following:

(1) Arithmetic IF — use the IF statement (8.1.2.4) or IF construct (8.1.2).

(2) Shared DO termination and termination on a statement other than END DO or CONTINUE — use an END DO or a CONTINUE statement for each DO statement.

(3) Alternate return — see B.2.1.

(4) Computed GO TO statement - see B.2.2.

(5) Statement functions - see B.2.3.

(6) DATA statements amongst executable statements - see B.2.4.

(7) Assumed length character functions - see B.2.5.

(8) Fixed form source - see B.2.6.

(9) CHARACTER* form of CHARACTER declaration - see B.2.7.

B.2.1 Alternate return

An alternate return introduces labels into an argument list to allow the called procedure to direct the execution of the caller upon return. The same effect can be achieved with a return code that is used in a CASE construct on return. This avoids an irregularity in the syntax and semantics of argument association. For example,

```
CALL SUBR_NAME (X, Y, Z, *100, *200, *300)
```

may be replaced by

```
CALL SUBR_NAME (X, Y, Z, RETURN_CODE)
SELECT CASE (RETURN_CODE)
   CASE (1)
      ...
   CASE (2)
      ...
```

```
   CASE (3)
      ...
   CASE DEFAULT
      ...
END SELECT
```

B.2.2 Computed GO TO statement

The computed GO TO has been superseded by the CASE construct, which is a generalized, easier to use and more efficient means of expressing the same computation.

B.2.3 Statement functions

Statement functions are subject to a number of non-intuitive restrictions and are a potential source of error since their syntax is easily confused with that of an assignment statement.

The internal function is a more generalized form of the statement function and completely supersedes it.

B.2.4 DATA statements among executables

The statement ordering rules of FORTRAN 66, and hence of FORTRAN 77 and Fortran 90 for compatibility, allowed DATA statements to appear anywhere in a program unit after the specification statements. The ability to position DATA statements amongst executable statements is very rarely used, is unnecessary and is a potential source of error.

B.2.5 Assumed character length functions

Assumed character length for functions is an irregularity in the language since elsewhere in Fortran the philosophy is that the attributes of a function result depend only on the actual arguments of the invocation and on any data accessible by the function through host or use association. Some uses of this facility can be replaced with an automatic character length function, where the length of the function result is declared in a specification expression. Other uses can be replaced by the use of a subroutine whose arguments correspond to the function result and the function arguments.

Note that dummy arguments of a function may be assumed character length.

B.2.6 Fixed form source

Fixed form source was designed when the principal machine-readable input medium for new programs was punched cards. Now that new and amended programs are generally entered via keyboards with screen displays, it is an unnecessary overhead, and is potentially error-prone, to have to locate positions 6, 7, or 72 on a line. Free form source was designed expressly for this more modern technology.

It is a simple matter for a software tool to convert from fixed to free form source.

B.2.7 CHARACTER* form of CHARACTER declaration

Fortran 90 had two different forms of specifying the length selector in CHARACTER declarations. The older form (CHARACTER*char-length) was an unnecessary redundancy.