Introduction

Exception handling is required for the development of robust and efficient numerical software. In particular, it is necessary in order to be able to write portable scientific libraries. In numerical Fortran programming, current practice is to employ whatever exception handling mechanisms are provided by the system/vendor. This clearly inhibits the production of fully portable numerical libraries and programs. It is particularly frustrating now that IEEE arithmetic (specified by IEEE 754-1985 *Standard for binary floating-point arithmetic*, also published as IEC 559:1989, *Binary floating-point arithmetic for microprocessor systems)* is so widely used, since built into it are the five conditions: overflow, invalid, divide-by-zero, underflow, and inexact. Our aim is to provide support for these conditions.

We have taken the opportunity to provide support for other aspects of the IEEE standard through a set of elemental functions that are applicable only to IEEE data types.

This proposal involves three standard modules:

- `IEEE_EXCEPTIONS` contains a derived type, some named constants of this type, and some simple procedures. They allow the flags to be tested, cleared, set, saved, or restored.
- `IEEE_ARITHMETIC` behaves as if it contained a `USE` statement for all of `IEEE_EXCEPTIONS` and provides support for other IEEE features through further derived types, named constants, and simple procedures.
- `IEEE_FEATURES` contains some named constants that permit the user to indicate which IEEE features are essential in the application. Some processors may execute more slowly when certain features are requested.

To facilitate maximum performance, each of the proposed functions does very little processing of arguments. In many cases, a processor may generate only a few inline machine code instructions rather than library calls.

In order to allow for the maximum number of processors to provide the maximum value to users, we do **not** require IEEE conformance. A vendor with no IEEE hardware need not provide these modules and any request by the user for any of them with a `USE` statement will give a compile-time diagnostic. A vendor whose hardware does not fully conform with the IEEE standard may be unable to provide certain features. In this case, a request for such a feature will give a compile-time diagnostic. Another possibility is that not all flags are supported or that the extent of support varies according to the kind type parameter. The user must utilize an inquiry function to determine if he or she can count on a specific feature of the IEEE standard.

Note that an implementor should avoid a macro implementation, as IEEE conformance is often controlled by compiler switches. A processor which offers a switch to turn off a facility should adjust the values returned for these inquiries. For example, a processor which allows gradual underflow to be turned off (replaced with flush to zero) should return false for `IEEE_SUPPORT_DENORMAL(X)` when a source file is processed with that option on. Naturally it should return true when that option is not in effect.

The most important use of a floating-point exception handling facility is to make possible the development of much more efficient software than is otherwise possible. The following ‘hypotenuse’ function, $\sqrt{x^2+y^2}$, illustrates the use of the facility in developing efficient software.

```
REAL FUNCTION HYPOT(X, Y)
! In rare circumstances this may lead to the signaling of IEEE_OVERFLOW
   USE, INTRINSIC :: IEEE_ARITHMETIC
   REAL X, Y
   REAL SCALED_X, SCALED_Y, SCALED_RESULT
   LOGICAL, DIMENSION(2) :: FLAGS
   TYPE (IEEE_FLAG_TYPE), PARAMETER, DIMENSION(2) :: &
          OUT_OF_RANGE = (/ IEEE_OVERFLOW, IEEE_UNDERFLOW /)
   INTRINSIC SQRT, ABS, EXPONENT, MAX, DIGITS, SCALE
! The processor clears the flags on entry
```

```
! Try a fast algorithm first
   HYPOT = SQRT( X**2 + Y**2 )
   CALL IEEE_GET_FLAG(OUT_OF_RANGE,FLAGS)
   IF ( ANY(FLAGS) ) THEN
      CALL IEEE_SET_FLAG(OUT_OF_RANGE,.FALSE.)
      IF ( X==0.0 .OR. Y==0.0 ) THEN
         HYPOT = ABS(X) + ABS(Y)
      ELSE IF ( 2*ABS(EXPONENT(X)-EXPONENT(Y)) > DIGITS(X)+1 ) THEN
         HYPOT = MAX( ABS(X), ABS(Y) )!  one of X and Y can be ignored
      ELSE     ! scale so that ABS(X) is near 1
         SCALED_X = SCALE( X, -EXPONENT(X) )
         SCALED_Y = SCALE( Y, -EXPONENT(X) )
         SCALED_RESULT = SQRT( SCALED_X**2 + SCALED_Y**2 )
         HYPOT = SCALE( SCALED_RESULT, EXPONENT(X) ) ! may cause overflow
      END IF
   END IF
! The processor resets any flag that was signaling on entry
END FUNCTION HYPOT
```

An attempt is made to evaluate this function directly in the fastest possible way. This will work almost every time, but if an exception occurs during this fast computation, a safe but slower way evaluates the function. This slower evaluation may involve scaling and unscaling, and in (very rare) extreme cases this unscaling can cause overflow (after all, the true result might overflow if `X` and `Y` are both near the overflow limit).

If the overflow or underflow flag is signaling on entry, it is reset on return by the processor, so that earlier exceptions are not lost.

Can all this be accomplished without the help of an exception handling facility? Yes, it can – in fact, the alternative code can do the job, but of course it is much less efficient. That's the point. The `HYPOT` function is special, in this respect, in that the normal and alternative codes try to accomplish the same task. This is not always the case. In fact, it very often happens that the alternative code concentrates on handling the exceptional cases and is not able to handle all of the non-exceptional cases. When this happens, a program which cannot take advantage of hardware flags could have a structure like the following:

```
if ( in the first exceptional region ) then
    handle this case
else if ( in the second exceptional region ) then
    handle this case
  :
else
    execute the normal code
end
```

But this is not only inefficient, it also inverts the logic of the computation. For other examples, see Hull, Fairgrieve and Tang (1994) and Demmel and Li (1994).

The code for the `HYPOT` function can be generalized in an obvious way to compute the Euclidean norm, $\sqrt{x_1^2+x_2^2+\ldots+x_n^2}$ of an n–vector; the generalization of the alternative code is not so obvious (though straightforward) and will be much slower relative to the normal code than is the case with the `HYPOT` function.

In connection with reliable computation, there is a need for intrinsic conditions further to those of the IEEE floating-point standard. Examples are:

- `INSUFFICIENT_STORAGE` for when the processor is unable to find sufficient storage to continue execution.
- `INTEGER_OVERFLOW` and `INTEGER_DIVIDE_BY_ZERO` for when an intrinsic integer operation has a very large result or has a zero denominator.

- `INTRINSIC` for when an intrinsic procedure has been unsuccessful.
- `SYSTEM_ERROR` for when a system error occurs.

This proposal has been designed to allow such enhancements in the future.

References

Demmel, J.W. and Li, X. (1994). Faster Numerical Algorithms via Exception Handling. IEEE Transactions on Computers, 43, no. 8, 983-992.

Hull, T.E., Fairgrieve, T.F., and Tang, T.P.T. (1994). Implementing complex elementary functions using exception handling. ACM Trans. Math. Software 20, 215-244.

Information technology – Programming languages – Fortran – Floating-point exception handling

1 General

1.1 Scope

This Technical Report specifies an extension of the programming language Fortran, specified by the international standard ISO/IEC 1539-1:1997. Its main aim is to provide support for the five exceptions of the IEEE standard for floating-point arithmetic, but it also provides support for other features of the IEEE standard. A processor is permitted to provide partial support and there are facilities for enquiring about which features are supported or requiring support of certain features.

Clause 2 of this Technical Report contains a technical description of the features. It provides an overview and does not include all the fine details. Clause 3 contains the editorial changes to the standard and thereby provides a complete definition.

1.2 Normative references

The following standards contain provisions which, through references in this text, constitute provisions of this Technical Report. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this Technical Report are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred applies. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 1539-1:1997, *Information technology – Programming Languages – Fortran – Part 1: Base language.*

IEC 559:1989, *Binary floating-point arithmetic for microprocessor systems.*

Since IEC 559:1989 was originally IEEE 754-1985 *Standard for binary floating-point arithmetic,* and is widely known by this name, we refer to it as the **IEEE standard** in this Technical Report.

2 Technical specification

2.1 The model

This proposal is based on the IEEE model with flags for the floating-point exceptions (invalid, overflow, divide-by-zero, underflow, inexact), a flag for the rounding mode (nearest, up, down, to-zero), and flags for whether halting occurs following exceptions. It is not necessary for the hardware to have any such flags (they may be simulated by software) or for it to support all the modes. Inquiry procedures are available to allow a program to determine the extent of support. Inquiries are in terms of reals, but the same level of support is provided for the corresponding complex kind.

Some hardware may be able to provide no support of these features or only partial support. It may execute faster with compiled code that does not support all the features. This proposal therefore involves three intrinsic modules. `IEEE_EXCEPTIONS` is for the exceptions and the minimum requirement is for the support of overflow and divide-by-zero for all kinds of real and complex data. `IEEE_ARITHMETIC` behaves as if it contained a `USE` statement for all of `IEEE_EXCEPTIONS` and provides support for other IEEE features. `IEEE_FEATURES` contains some named constants that permit the user to indicate which features are essential in the application. A program is required to fail if a requested feature is not available. The modules contain five derived types (subclause 2.3), named constants to control the level of support (subclause 2.4), and a collection of procedures (subclauses 2.5 to 2.10). None of the procedures is permitted as an actual argument.

2.2 The USE statement for an intrinsic module

New syntax on the USE statement provides control over whether it is intended to access an intrinsic module:

```
USE, INTRINSIC :: IEEE_ARITHMETIC
```

or not:

```
USE, NON_INTRINSIC :: MY_IEEE_ARITHMETIC
```

The `INTRINSIC` statement is not extended. For the old form:

```
USE IEEE_ARITHMETIC
```

the processor looks first for a non-intrinsic module.

2.3 The derived types and data objects

The module `IEEE_EXCEPTIONS` contains the derived types:

- `IEEE_FLAG_TYPE`, for identifying a particular exception flag. Its only possible values are those of named constants defined in the module: `IEEE_INVALID`, `IEEE_OVERFLOW`, `IEEE_DIVIDE_BY_ZERO`, `IEEE_UNDERFLOW`, and `IEEE_INEXACT`. The modules also contains the named array constants
  ```
  IEEE_USUAL = (/IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, IEEE_INVALID/)
  ```
  and
  ```
  IEEE_ALL = (/IEEE_USUAL, IEEE_UNDERFLOW, IEEE_INEXACT/)
  ```
- `IEEE_STATUS_TYPE`, for saving the current floating point status.

The module `IEEE_ARITHMETIC` contains the derived types:

- `IEEE_CLASS_TYPE`, for identifying a class of floating-point values. Its only possible values are those of named constants defined in the module:
  `IEEE_SIGNALING_NAN`, `IEEE_QUIET_NAN`, `IEEE_NEGATIVE_INF`, `IEEE_NEGATIVE_NORMAL`, `IEEE_NEGATIVE_DENORMAL`, `IEEE_NEGATIVE_ZERO`, `IEEE_POSITIVE_ZERO`, `IEEE_POSITIVE_DENORMAL`, `IEEE_POSITIVE_NORMAL`, and `IEEE_POSITIVE_INF`.

- IEEE_ROUND_TYPE, for identifying a particular rounding mode. Its only possible values are those of named constants defined in the module: IEEE_NEAREST, IEEE_TO_ZERO, IEEE_UP, and IEEE_DOWN for the IEEE modes; and IEEE_OTHER for any other mode.
- The elemental operator == for two values of one of these types to return .TRUE. if the values are the same and .FALSE. otherwise.
- The elemental operator /= for two values of one of these types to return .TRUE. if the values differ and .FALSE. otherwise.

The module IEEE_FEATURES contains the derived type:

- IEEE_FEATURES_TYPE, for expressing the need for particular IEEE features. Its only possible values are those of named constants defined in the module: IEEE_DATATYPE, IEEE_DENORMAL, IEEE_DIVIDE, IEEE_HALTING, IEEE_INEXACT_FLAG, IEEE_INF, IEEE_INVALID_FLAG, IEEE_NAN, IEEE_ROUNDING, IEEE_SQRT, and IEEE_UNDERFLOW_FLAG.

2.4 The level of support

When IEEE_EXCEPTIONS or IEEE_ARITHMETIC is accessible, IEEE_OVERFLOW and IEEE_DIVIDE_BY_ZERO are supported in the scoping unit for all kinds of real and complex data. Which other exceptions are supported may be determined by the function IEEE_SUPPORT_FLAG, see subclause 2.6. Whether control of halting is supported may be determined by the function IEEE_SUPPORT_HALTING. The extent of support of the other exceptions may be influenced by the accessibility of the named constants IEEE_INEXACT_FLAG, IEEE_INVALID_FLAG, and IEEE_UNDERFLOW_FLAG of the module IEEE_FEATURES. If a scoping unit has access to IEEE_UNDERFLOW_FLAG of IEEE_FEATURES, the scoping unit must support underflow and return true from IEEE_SUPPORT_FLAG(IEEE_UNDERFLOW,X) for at least one kind of real. Similarly, if IEEE_INEXACT_FLAG or IEEE_INVALID_FLAG is accessible, the scoping unit must support the exception and return true from the corresponding inquiry for at least one kind of real. Also, if IEEE_HALTING is accessible, the scoping unit must support control of halting and return true from IEEE_SUPPORT_HALTING(FLAG) for the flag.

If a scoping unit does not access IEEE_EXCEPTIONS or IEEE_ARITHMETIC, the level of support is processor dependent, and need not include support for any exceptions. If a flag is signaling on entry to such a scoping unit, the processor ensures that it is signaling on exit. If a flag is quiet on entry to such a scoping unit, whether it is signaling on exit is processor dependent.

For processors with IEEE arithmetic, further IEEE support is available through the module IEEE_ARITHMETIC. The extent of support may be influenced by the accessibility of the named constants of the module IEEE_FEATURES. If a scoping unit has access to IEEE_DATATYPE of IEEE_FEATURES, the scoping unit must support IEEE arithmetic and return true from IEEE_SUPPORT_DATATYPE(X) (see subclause 2.6) for at least one kind of real. Similarly, if IEEE_DENORMAL, IEEE_DIVIDE, IEEE_INF, IEEE_NAN, IEEE_ROUNDING, or IEEE_SQRT is accessible, the scoping unit must support the feature and return true from the corresponding inquiry function for at least one kind of real. In the case of IEEE_ROUNDING, it must return true for all the rounding modes IEEE_NEAREST, IEEE_TO_ZERO, IEEE_UP, and IEEE_DOWN.

Execution may be slowed on some processors by the support of some features. If IEEE_EXCEPTIONS or IEEE_ARITHMETIC is accessed but IEEE_FEATURES is not accessed, the vendor is free to choose which subset to support. The processor's fullest support is provided when all of IEEE_FEATURES is accessed:

```
USE IEEE_ARITHMETIC; USE IEEE_FEATURES
```

but execution may then be slowed by the presence of a feature that is not needed. In all cases, the extent of support may be determined by the inquiry functions of subclause 2.6.

If a flag is signaling on entry to a procedure, the processor will set it to quiet on entry and restore it to signaling on return.

If a flag is quiet on entry to a procedure with access to `IEEE_EXCEPTIONS` or `IEEE_ARITHMETIC` and is signaling on return, the processor will not restore it to quiet.

In a procedure, the processor ensures that the flags for halting have the same values on return as on entry.

In a procedure, the processor ensures that the flags for rounding have the same values on return as on entry.

2.5 The exception flags

The flags are initially quiet and signal when an exception occurs. The value of a flag is determined by the elemental subroutine

```
IEEE_GET_FLAG (FLAG,FLAG_VALUE)
```

where `FLAG` is of type `IEEE_FLAG_TYPE` and `FLAG_VALUE` is of type default `LOGICAL`. Being elemental allows an array of flag values to be obtained at once and obviates the need for a list of flags.

Flag values may be assigned by the elemental subroutine

```
IEEE_SET_FLAG (FLAG,FLAG_VALUE)
```

An exception must not signal if this could arise only during execution of an operation further to those required or permitted by the standard. For example, the statement

```
IF (F(X)>0.0) Y = 1.0/Z
```

must not signal `IEEE_DIVIDE_BY_ZERO` when both `F(X)` and `Z` are zero and the statement

```
WHERE(A>0.0) A = 1.0/A
```

must not signal `IEEE_DIVIDE_BY_ZERO.` On the other hand, when `X` has the value 1.0 and `Y` has the value 0.0, the expression

```
X>0.00001 .OR. X/Y>0.00001
```

is permitted to cause the signaling of `IEEE_DIVIDE_BY_ZERO`.

2.6 Inquiry functions for the features supported

The module `IEEE_EXCEPTIONS` contains the following inquiry functions:

- `IEEE_SUPPORT_FLAG(FLAG[,X])` True if the processor supports an exception flag for all reals (`X` absent) or for reals of the same kind type parameter as the argument `X`.
- `IEEE_SUPPORT_HALTING(FLAG)` True if the processor supports the ability to control during program execution whether to abort or continue execution after an exception.

The module `IEEE_ARITHMETIC` contains the following inquiry functions:

- `IEEE_SUPPORT_DATATYPE([X])` True if the processor supports IEEE arithmetic for all reals (`X` absent) or for reals of the same kind type parameter as the argument `X`. Here support means employing an IEEE data format and performing the operations of `+`, `–`, and `*` as in the IEEE standard whenever the operands and result all have normal values.
- `IEEE_SUPPORT_DENORMAL([X])` True if the processor supports the IEEE denormalized numbers for all reals (`X` absent) or for reals of the same kind type parameter as the argument `X`.
- `IEEE_SUPPORT_DIVIDE([X])` True if the processor supports divide with the accuracy specified by the IEEE standard for all reals (`X` absent) or for reals of the same kind type parameter as the argument `X`.
- `IEEE_SUPPORT_INF([X])` True if the processor supports the IEEE infinity facility for all reals (`X`

absent) or for reals of the same kind type parameter as the argument X.

- IEEE_SUPPORT_NAN([X]) True if the processor supports the IEEE Not-A-Number facility for all reals (X absent) or for reals of the same kind type parameter as the argument X.
- IEEE_SUPPORT_ROUNDING(ROUND_VALUE[,X]) True if the processor supports a particular rounding mode for all reals (X absent) or for reals of the same kind type parameter as the argument X. Here, support includes the ability to change the mode by

```
CALL IEEE_SET_ROUNDING_MODE(ROUND_VALUE)
```

- IEEE_SUPPORT_SQRT([X]) True if the processor supports IEEE square root for all reals (X absent) or for reals of the same kind type parameter as the argument X.
- IEEE_SUPPORT_STANDARD([X]) True if the processor supports all the IEEE facilities defined in this Technical Report for all reals (X absent) or for reals of the same kind type parameter as the argument X.

2.7 Elemental functions

The module IEEE_ARITHMETIC contains the following elemental functions for reals X and Y for which IEEE_SUPPORT_DATATYPE(X) and IEEE_SUPPORT_DATATYPE(Y) are true:

- IEEE_CLASS(X) Returns the IEEE class (see subclause 2.3 for the possible values).
- IEEE_COPY_SIGN(X,Y) IEEE copysign function, that is X with the sign of Y.
- IEEE_IS_FINITE(X) IEEE finite function. True if IEEE_CLASS(X) has one of the values IEEE_NEGATIVE_NORMAL, IEEE_NEGATIVE_DENORMAL, IEEE_NEGATIVE_ZERO, IEEE_POSITIVE_ZERO, IEEE_POSITIVE_DENORMAL, IEEE_POSITIVE_NORMAL.
- IEEE_IS_NAN(X) True if the value is IEEE Not-a-Number.
- IEEE_IS_NEGATIVE(X) True if the value is negative (including negative zero).
- IEEE_IS_NORMAL(X) True if the value is a normal number.
- IEEE_LOGB(X) IEEE *logb* function, that is, the unbiased exponent of X.
- IEEE_NEXT_AFTER(X,Y) Returns the next representable neighbor of X in the direction toward Y.
- IEEE_REM(X,Y) The IEEE REM function, that is X – Y*N, where N is the integer nearest to the exact value X/Y.
- IEEE_RINT(X) Round to an integer value according to the current rounding mode.
- IEEE_SCALB (X,I) Returns 2^{I}X.
- IEEE_UNORDERED(X,Y) IEEE unordered function. True if X or Y is a NaN and false otherwise.
- IEEE_VALUE(X,CLASS) Generate a value of a given IEEE class. The value of CLASS is permitted to be
    - IEEE_SIGNALING_NAN or IEEE_QUIET_NAN if IEEE_SUPPORT_NAN(X) has the value true,
    - IEEE_NEGATIVE_INF or IEEE_POSITIVE_INF if IEEE_SUPPORT_INF(X) has the value true,
    - IEEE_NEGATIVE_DENORMAL or IEEE_POSITIVE_DENORMAL if IEEE_SUPPORT_DENORMAL(X) has the value true,
    - IEEE_NEGATIVE_NORMAL, IEEE_NEGATIVE_ZERO, IEEE_POSITIVE_ZERO or IEEE_POSITIVE_NORMAL.

Although in most cases the value is processor dependent, the value does not vary between invocations for any particular X kind type parameter and CLASS value.

2.8 Elemental subroutines

The module IEEE_EXCEPTIONS contains the following elemental subroutines:

- IEEE_GET_FLAG(FLAG,FLAG_VALUE) Get an exception flag.
- IEEE_GET_HALTING_MODE(FLAG,HALTING) Get halting mode for an exception. The initial halting mode is processor dependent. Halting is not necessarily immediate, but normal processing does not continue.
- IEEE_SET_FLAG(FLAG,FLAG_VALUE) Set an exception flag.
- IEEE_SET_HALTING_MODE(FLAG,HALTING) Controls continuation or halting on exceptions.

2.9 Non-elemental subroutines

The module IEEE_EXCEPTIONS contains the following non-elemental subroutines:

- IEEE_GET_STATUS(STATUS_VALUE) Get the current values of the set of flags that define the current state of the floating point environment. STATUS_VALUE is of type IEEE_STATUS_TYPE.
- IEEE_SET_STATUS(STATUS_VALUE) Restore the values of the set of flags that define the current state of the floating point environment (usually the floating point status register). STATUS_VALUE is of type IEEE_STATUS_TYPE and has been set by a call of IEEE_GET_STATUS.

The module IEEE_ARITHMETIC contains the following non-elemental subroutines:

- IEEE_GET_ROUNDING_MODE(ROUND_VALUE) Get the current IEEE rounding mode. ROUND_VALUE is of type IEEE_ROUND_TYPE.
- IEEE_SET_ROUNDING_MODE(ROUND_VALUE) Set the current IEEE rounding mode. ROUND_VALUE is of type IEEE_ROUND_TYPE. If this is invoked, IEEE_SUPPORT_ROUNDING(ROUND_VALUE,X) must be true for any X such that IEEE_SUPPORT_DATATYPE(X) is true.

2.10 Transformational function

The module IEEE_ARITHMETIC contains the following transformational function:

- IEEE_SELECTED_REAL_KIND([P,][R]) As for SELECTED_REAL_KIND but gives an IEEE kind.

3 Edits to the standard (ISO/IEC 1539-1:1997)

xvi/16. Add 'A module may be intrinsic (defined by the standard) or nonintrinsic (defined by Fortran code).'

19/6. After 'procedures,' add 'modules,'.

94/11+. Add new item:

(5a) A reference to the transformational function `IEEE_SELECTED_REAL_KIND` from the intrinsic module `IEEE_ARITHMETIC`, where each argument is an initialization expression.

131/33. Add: 'If any exception (15) is signaling, the processor shall issue a warning on the unit identified by `*` in a `WRITE` statement, indicating which exceptions are signaling.'.

186/17. Add 'An **intrinsic module** is defined by the standard. A **nonintrinsic module** is defined by Fortran code.'

186/17+ Add paragraph:
Procedures and types defined in an intrinsic module are not themselves intrinsic.

187/22-23. Change to

R1107 *use-stmt* **is** USE [[, *module-nature*] ::] *module-name* [, *rename-list*]
or USE [[, *module-nature*] ::] *module-name,* ONLY: [*only-list*]

R1107a *module-nature* **is** INTRINSIC
or NON_INTRINSIC

Constraint: If *module-nature* is INTRINSIC, *module-name* shall be the name of an intrinsic module.

Constraint: If *module-nature* is NON_INTRINSIC, *module-name* shall be the name of a nonintrinsic module.

187/31+. Add 'A *use-stmt* without a *module-nature* provides access either to an intrinsic or to a nonintrinsic module. If the *module-name* is the name of both an intrinsic and a nonintrinsic module, the nonintrinsic module is accessed.'

228/36. Change '.' to ', unless the intrinsic module IEEE_ARITHMETIC (clause 15) is accessible and there is support for an infinite or a NaN result, as appropriate. If an infinite result is returned, the flag IEEE_OVERFLOW or IEEE_DIVIDE_BY_ZERO shall signal; if a NaN result is returned, the flag IEEE_INVALID shall signal. If all results are normal, these flags must have the same status as when the intrinsic procedure was invoked.'

297/10. Change '*intrinsic function*' to '*function* that is either *intrinsic* or is defined in an intrinsic *module* and'.

297/25. Change 'and *procedures'* to '*procedures,* and *modules'*.

297/26. After 'and' add 'that'.

297/27. Add 'Procedures and types defined in an intrinsic module are not themselves intrinsic.'.

301/1 Change '*intrinsic function*' to '*function* that is either *intrinsic* or is defined in an intrinsic *module* and'.

292+. Add

15. Exceptions and IEEE arithmetic

The modules IEEE_EXCEPTIONS, IEEE_ARITHMETIC, and IEEE_FEATURES provide support for exceptions and IEEE arithmetic. Whether the modules are provided is processor dependent. If the module IEEE_FEATURES is provided, which of the named constants defined by this standard are included is processor dependent. The module IEEE_ARITHMETIC behaves as if it contained a USE statement for IEEE_EXCEPTIONS and everything that is public in IEEE_EXCEPTIONS is public in IEEE_ARITHMETIC.

When IEEE_EXCEPTIONS or IEEE_ARITHMETIC is accessible, IEEE_OVERFLOW and

IEEE_DIVIDE_BY_ZERO are supported in the scoping unit for all kinds of real and complex data. Which other exceptions are supported may be determined by the function IEEE_SUPPORT_FLAG, see subclause 15.9, and whether control of halting is supported may be determined by the function IEEE_SUPPORT_HALTING. The extent of support of the other exceptions may be influenced by the accessibility of the named constants IEEE_INEXACT_FLAG, IEEE_INVALID_FLAG, and IEEE_UNDERFLOW_FLAG of the module IEEE_FEATURES. If a scoping unit has access to IEEE_UNDERFLOW_FLAG of IEEE_FEATURES, the scoping unit must support underflow and return true from IEEE_SUPPORT_FLAG(IEEE_UNDERFLOW, X) for at least one kind of real. Similarly, if IEEE_INEXACT_FLAG or IEEE_INVALID_FLAG is accessible, the scoping unit must support the exception and return true from the corresponding inquiry for at least one kind of real. Also, if IEEE_HALTING is accessible, the scoping unit must support control of halting and return true from IEEE_SUPPORT_HALTING(FLAG) for the flag.

If a scoping unit does not access IEEE_EXCEPTIONS or IEEE_ARITHMETIC, the level of support is processor dependent, and need not include support for any exceptions. If a flag is signaling on entry to such a scoping unit, the processor ensures that it is signaling on exit. If a flag is quiet on entry to such a scoping unit, whether it is signaling on exit is processor dependent.

For processors with IEEE arithmetic, further IEEE support is available through the module IEEE_ARITHMETIC. The extent of support may be influenced by the accessibility of the named constants of the module IEEE_FEATURES. If a scoping unit has access to IEEE_DATATYPE of IEEE_FEATURES, the scoping unit must support IEEE arithmetic and return true from IEEE_SUPPORT_DATATYPE(X) (see subclause 15.9) for at least one kind of real. Similarly, if IEEE_DENORMAL, IEEE_DIVIDE, IEEE_INF, IEEE_NAN, IEEE_ROUNDING, or IEEE_SQRT is accessible, the scoping unit must support the feature and return true from the corresponding inquiry function for at least one kind of real. In the case of IEEE_ROUNDING, it must return true for all the rounding modes IEEE_NEAREST, IEEE_TO_ZERO, IEEE_UP, and IEEE_DOWN.

Execution may be slowed on some processors by the support of some features. If IEEE_EXCEPTIONS or IEEE_ARITHMETIC is accessed but IEEE_FEATURES is not accessed, the vendor is free to choose which subset to support. The processor's fullest support is provided when all of IEEE_FEATURES is accessed:

```
USE IEEE_ARITHMETIC; USE IEEE_FEATURES
```

but execution may then be slowed by the presence of a feature that is not needed. In all cases, the extent of support may be determined by the inquiry functions.

Note: The types and procedures defined in these modules are not themselves intrinsic.

15.1 Derived data types defined in the module

The modules IEEE_EXCEPTIONS, IEEE_ARITHMETIC, and IEEE_FEATURES contain five derived types, whose components are private.

The module IEEE_EXCEPTIONS contains:

- IEEE_FLAG_TYPE, for identifying a particular exception flag. Its only possible values are those of named constants defined in the modules: IEEE_INVALID, IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, IEEE_UNDERFLOW, and IEEE_INEXACT. The modules also contains the array named constants
  IEEE_USUAL = (/ IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, IEEE_INVALID /)
  and IEEE_ALL = (/ IEEE_USUAL, IEEE_UNDERFLOW, IEEE_INEXACT /).
- IEEE_STATUS_TYPE, for saving the current floating point status.

The module IEEE_ARITHMETIC contains:

- IEEE_CLASS_TYPE, for identifying a class of floating-point values. Its only possible values are those of named constants defined in the module: IEEE_SIGNALING_NAN, IEEE_QUIET_NAN, IEEE_NEGATIVE_INF, IEEE_NEGATIVE_NORMAL, IEEE_NEGATIVE_DENORMAL, IEEE_NEGATIVE_ZERO, IEEE_POSITIVE_ZERO, IEEE_POSITIVE_DENORMAL, IEEE_POSITIVE_NORMAL, IEEE_POSITIVE_INF.
- IEEE_ROUND_TYPE, for identifying a particular rounding mode. Its only possible values are those of named constants defined in the module: IEEE_NEAREST, IEEE_TO_ZERO, IEEE_UP, and IEEE_DOWN for the IEEE modes; and IEEE_OTHER for any other mode.
- The elemental operator == for two values of one of these types to return .TRUE. if the values are the same and .FALSE. otherwise.
- The elemental operator /= for two values of one of these types to return .TRUE. if the values differ and .FALSE. otherwise.

The module IEEE_FEATURES contains:

- IEEE_FEATURES_TYPE, for expressing the need for particular IEEE features. Its only possible values are those of named constants defined in the module: IEEE_DATATYPE, IEEE_DENORMAL, IEEE_DIVIDE, IEEE_HALTING, IEEE_INEXACT_FLAG, IEEE_INF, IEEE_INVALID_FLAG, IEEE_NAN, IEEE_ROUNDING, IEEE_SQRT, and IEEE_UNDERFLOW_FLAG.

15.2 The exceptions

The exceptions are:

- IEEE_OVERFLOW
  This exception occurs when the result for an intrinsic real operation or assignment has an absolute value greater than a processor-dependent limit, or the real or imaginary part of the result for an intrinsic complex operation or assignment has an absolute value greater than a processor-dependent limit.
- IEEE_DIVIDE_BY_ZERO
  This exception occurs when a real or complex division has a nonzero numerator and a zero denominator.
- IEEE_INVALID
  This exception occurs when a real or complex operation or assignment is invalid; examples are SQRT(X) when X is real and has a nonzero negative value, and conversion to an integer (by assignment or an intrinsic procedure) when the result is too large to be representable.
- IEEE_UNDERFLOW
  This exception occurs when the result for an intrinsic real operation or assignment has an absolute value less than a processor-dependent limit and loss of accuracy is detected, or the real or imaginary part of the result for an intrinsic complex operation or assignment has an absolute value less than a processor-dependent limit and loss of accuracy is detected.
- IEEE_INEXACT
  This exception occurs when the result of a real or complex operation or assignment is not exact.

Each exception has a flag whose value is either quiet or signaling. The value may be determined by the function IEEE_GET_FLAG. Its initial value is quiet and it signals when the associated exception occurs. Its status may also be changed by the subroutine IEEE_SET_FLAG or the subroutine IEEE_SET_STATUS. Once signaling, it remains signaling unless set quiet by an invocation of the subroutine IEEE_SET_FLAG or the subroutine IEEE_SET_STATUS.

If a flag is signaling on entry to a procedure, the processor will set it to quiet on entry and restore it to signaling on return.

Evaluation of a specification expression may cause an exception to signal.

In a scoping unit that has access to IEEE_EXCEPTIONS or IEEE_ARITHMETIC, if an intrinsic procedure executes normally, the values of the flags IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, and IEEE_INVALID shall be as on entry to the procedure, even if one or more signals during the calculation. If a real or complex result is too large for the intrinsic to handle, IEEE_OVERFLOW may signal. If a real or complex result is a NaN because of an invalid operation (for example, LOG(–1.0)), IEEE_INVALID may signal. Similar rules apply to format processing and to intrinsic operations: no signaling flag shall be set quiet and no quiet flag shall be set signaling because of an intermediate calculation that does not affect the result.

> Note: An implementation may provide alternative versions of an intrinsic procedure; a practical example of such alternatives might be one version suitable for a call from a scoping unit with access to IEEE_EXCEPTIONS or IEEE_ARITHMETIC and one for other cases.

In a sequence of statements that contains no invocations of IEEE_GET_FLAG, IEEE_SET_FLAG, IEEE_GET_STATUS, IEEE_SET_HALTING, or IEEE_SET_STATUS, if the execution of an operation would cause an exception to signal but after execution of the sequence no value of a variable depends on the operation, whether the exception is signaling is processor dependent. For example, when Y has the value zero, whether the code

```
X = 1.0/Y
X = 3.0
```

signals IEEE_DIVIDE_BY_ZERO is processor dependent. Another example is the following:

```
REAL, PARAMETER :: X=0.0, Y=6.0
:
IF (1.0/X == Y) PRINT *,'Hello world'
```

where the processor is permitted to discard the IF statement since the logical expression can never be true and no value of a variable depends on it.

An exception must not signal if this could arise only during execution of a operation further to those required or permitted by the standard. For example, the statement

```
IF (F(X)>0.0) Y = 1.0/Z
```

must not signal IEEE_DIVIDE_BY_ZERO when both F(X) and Z are zero and the statement

```
WHERE(A>0.0) A = 1.0/A
```

must not signal IEEE_DIVIDE_BY_ZERO. On the other hand, when X has the value 1.0 and Y has the value 0.0, the expression

```
X>0.00001 .OR. X/Y>0.00001
```

is permitted to cause the signaling of IEEE_DIVIDE_BY_ZERO.

The processor need not support IEEE_INVALID, IEEE_UNDERFLOW, and IEEE_INEXACT. If an exception is not supported, its flag is always quiet. The function IEEE_SUPPORT_FLAG may be used to inquire whether a particular flag is supported. If IEEE_INVALID is supported, it signals in the case of conversion to an integer (by assignment or an intrinsic procedure) if the result is too large to be representable.

15.3 The rounding modes

IEEE 754-1985 (also IEC 559:1989) specifies four rounding modes:

- IEEE_NEAREST rounds the exact result to the nearest representable value.
- IEEE_TO_ZERO rounds the exact result towards zero to the next representable value.

- IEEE_UP rounds the exact result towards +infinity to the next representable value.
- IEEE_DOWN rounds the exact result towards –infinity to the next representable value.

The function IEEE_GET_ROUNDING_MODE may be used to inquire which rounding mode is in operation. Its value is one of the above four or IEEE_OTHER if the rounding mode does not conform to IEEE 754-1985.

If the processor supports the alteration of the rounding mode during execution, the subroutine IEEE_SET_ROUNDING_MODE may be used to alter it. The function IEEE_SUPPORT_ROUNDING may be used to inquire whether this facility is available for a particular mode.

In a procedure other than IEEE_SET_ROUNDING_MODE, the processor shall not change the rounding mode on entry, and on return shall ensure that the rounding mode is the same as it was on entry.

Note. Within a program, all literal constants that have the same form have the same value (4.1.2). Therefore, the value of a literal constant is not affected by the rounding mode.

15.4 Halting

Some processors allow control during program execution of whether to abort or continue execution after an exception. Such control is exercised by invocation of the subroutine IEEE_SET_HALTING_MODE. Halting is not precise and may occur any time after the exception has occurred. The function IEEE_SUPPORT_HALTING may be used to inquire whether this facility is available. The initial halting mode is processor dependent.

In a procedure other than IEEE_SET_HALTING_MODE, the processor shall not change the halting mode on entry, and on return shall ensure that the halting mode is the same as it was on entry.

15.5 The floating point status

The values of all the supported flags for exceptions, rounding mode, and halting may be saved in a scalar variable of type TYPE(IEEE_STATUS_TYPE) with the function IEEE_GET_STATUS and restored with the subroutine IEEE_SET_STATUS. There are no facilities for finding the values of particular flags held within such a variable.

> Note. Some processors hold all these flags in a floating point status register that can be saved and restored as a whole much faster than all individual flags can be saved and restored. These procedures are provided to exploit this feature.

15.6 Exceptional values

IEEE 754-1985 specifies the following exceptional floating point values:

- Denormalized values have very small absolute values and lowered precision.
- Infinite values (+Inf and –Inf) are created by overflow or division by zero.
- Not-a-Number (NaN) values are undefined values or values created by an invalid operation.

In this standard, the term **normal** is used for values that are not in one of these exceptional classes.

The functions IEEE_IS_FINITE, IEEE_IS_NAN, IEEE_IS_NEGATIVE, and IEEE_IS_NORMAL are provided to test whether a value is finite, NaN, negative, or normal. The function IEEE_VALUE is provided to generate an IEEE number of any class, including an infinity or a NaN. The functions IEEE_SUPPORT_DENORMAL, IEEE_SUPPORT_DIVIDE, IEEE_SUPPORT_INF, and IEEE_SUPPORT_NAN may be used to inquire whether this facility is available for a particular kind of real.

15.7 IEEE arithmetic

The function IEEE_SUPPORT_DATATYPE may be used to inquire whether IEEE arithmetic is available for a particular kind of real. Complete conformance with the IEEE standard is not required, but the normalized numbers must be exactly those of IEEE single or IEEE double; the arithmetic operators must be implemented with at least one of the IEEE rounding modes; and the functions copysign, scalb, logb, nextafter, REM, and unordered must be provided by the functions IEEE_COPY_SIGN, IEEE_SCALB, IEEE_LOGB, IEEE_NEXT_AFTER, IEEE_REM, and IEEE_UNORDERED. The inquiry function IEEE_SUPPORT_DIVIDE is provided to inquire whether the processor supports divide with the accuracy specified by the IEEE standard. For each of the other arithmetic operators and for each implemented IEEE rounding mode, the result shall be as specified in the IEEE standard whenever the operands and IEEE result are normalized.

The inquiry function IEEE_SUPPORT_NAN is provided to inquire whether the processor supports IEEE NaNs. Where these are supported, their behaviour for unary and binary operations, including those defined by intrinsic functions and by functions in intrinsic modules, is as specified in the IEEE standard.

The inquiry function IEEE_SUPPORT_INF is provided to inquire whether the processor supports IEEE infinities. Where these are supported, their behaviour for unary and binary operations, including those defined by intrinsic functions and by functions in intrinsic modules, is as specified in the IEEE standard.

IEEE 754-1985 specifies a square root function that returns –0.0 for the square root of –0.0. The function IEEE_SUPPORT_SQRT may be used to inquire whether SQRT is implemented in accord with the IEEE standard for a particular kind of real.

The inquiry function IEEE_SUPPORT_STANDARD is provided to inquire whether the processor supports all the IEEE facilities defined in this standard for a particular kind of real.

15.8 Tables of the procedures

In this subclause, the procedures defined in the modules are tabulated with the names of their arguments and a short description.

15.8.1 Inquiry functions

The module IEEE_EXCEPTIONS contains the following inquiry functions:

- IEEE_SUPPORT_FLAG(FLAG [, X]) Inquire if the processor supports an exception.
- IEEE_SUPPORT_HALTING(FLAG) Inquire if the processor supports control of halting after an exception.

The module IEEE_ARITHMETIC contains the following inquiry functions:

- IEEE_SUPPORT_DATATYPE([X]) Inquire if the processor supports IEEE arithmetic.
- IEEE_SUPPORT_DENORMAL([X]) Inquire if the processor supports denormalized numbers.
- IEEE_SUPPORT_DIVIDE([X]) Inquire if the processor supports divide with the accuracy specified by the IEEE standard.
- IEEE_SUPPORT_INF([X]) Inquire if processor supports the IEEE infinity.
- IEEE_SUPPORT_NAN([X]) Inquire if processor supports the IEEE Not-A-Number.
- IEEE_SUPPORT_ROUNDING(ROUND_VALUE [, X]) Inquire if processor supports a particular rounding mode.
- IEEE_SUPPORT_SQRT([X]) Inquire if the processor supports IEEE square root.

- IEEE_SUPPORT_STANDARD([X]) Inquire if processor supports all IEEE facilities.

15.8.2 Elemental functions

The module IEEE_ARITHMETIC contains the following elemental functions for reals X and Y for which IEEE_SUPPORT_DATATYPE(X) and IEEE_SUPPORT_DATATYPE(Y) are true:

- IEEE_CLASS(X) IEEE class.
- IEEE_COPY_SIGN(X,Y) IEEE copysign function.
- IEEE_IS_FINITE(X) Determine if value is finite.
- IEEE_IS_NAN(X) Determine if value is IEEE Not-a-Number.
- IEEE_IS_NORMAL(X) Whether a value is normal, that is, neither an Infinity, a NaN, nor denormalized.
- IEEE_LOGB(X) Unbiased exponent in the IEEE floating point format.
- IEEE_NEXT_AFTER(X,Y) Returns the next representable neighbor of X in the direction toward Y.
- IEEE_REM(X,Y) The IEEE REM function, that is X – Y*N, where N is the integer nearest to the exact value X/Y.
- IEEE_RINT(X) Round to an integer value according to the current rounding mode.
- IEEE_SCALB(X,I) Returns $2^{I}X$.
- IEEE_UNORDERED(X,Y) IEEE unordered function. True if X or Y is a NaN and false otherwise.
- IEEE_VALUE(X, CLASS) Generate an IEEE value.

  Note: If X or Y has a value that is an infinity or a NaN, the result shall be consistent with the general rules in sections 6.1 and 6.2 of the IEEE standard. For example, the result for an infinity shall be constructed as the limiting case of the result with a value of arbitrarily large magnitude, when such a limit exists.

15.8.3 Kind function

The module IEEE_ARITHMETIC contains the following transformational function:

- IEEE_SELECTED_REAL_KIND ([P,][R]) Kind type parameter value for an IEEE real with given precision and range.

15.8.4 Elemental subroutines

The module IEEE_EXCEPTIONS contains the following elemental subroutines:

- IEEE_GET_FLAG(FLAG,FLAG_VALUE) Get an exception flag.
- IEEE_GET_HALTING_MODE(FLAG, HALTING) Get halting mode for an exception.
- IEEE_SET_FLAG(FLAG,FLAG_VALUE) Set an exception flag.
- IEEE_SET_HALTING_MODE(FLAG,HALTING) Controls continuation or halting on exceptions.

15.8.5 Non-elemental subroutines

The module IEEE_EXCEPTIONS contains the following non-elemental subroutines:

- IEEE_GET_STATUS(STATUS_VALUE) Get the current state of the floating point environment.
- IEEE_SET_STATUS(STATUS_VALUE) Restore the state of the floating point environment.

The module IEEE_ARITHMETIC contains the following non-elemental subroutines:

- IEEE_GET_ROUNDING_MODE(ROUND_VALUE) Get the current IEEE rounding mode.
- IEEE_SET_ROUNDING_MODE(ROUND_VALUE) Set the current IEEE rounding mode.

15.9 Specifications of the procedures

In this section, the procedures are described in detail. The procedure names are generic and are not specific. All the functions are pure. In the examples, it is assumed that the processor supports IEEE arithmetic for default real.

Note: It is intended that processors should not check a condition given in a paragragh labeled **Restriction** at compile time, but rather should rely on the programmer writing code such as

```
IF (IEEE_SUPPORT_DATATYPE(X)) THEN
   C = IEEE_CLASS(X)
ELSE
   :
END IF
```

to avoid a call ever being made on a processor for which the condition is violated.

15.9.1 IEEE_CLASS (X)

Description. IEEE class function.

Class. Elemental function.

Argument. X shall be of type real.

Restriction. A program is prohibited from invoking this procedure if IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. TYPE(IEEE_CLASS_TYPE).

Result Value. The result value is one of: IEEE_SIGNALING_NAN, IEEE_QUIET_NAN, IEEE_NEGATIVE_INF, IEEE_NEGATIVE_NORMAL, IEEE_NEGATIVE_DENORMAL, IEEE_NEGATIVE_ZERO, IEEE_POSITIVE_ZERO, IEEE_POSITIVE_DENORMAL, IEEE_POSITIVE_NORMAL, IEEE_POSITIVE_INF. Neither of the values IEEE_SIGNALING_NAN and IEEE_QUIET_NAN shall be returned unless IEEE_SUPPORT_NAN(X) has the value true. Neither of the values IEEE_NEGATIVE_INF and IEEE_POSITIVE_INF shall be returned unless IEEE_SUPPORT_INF(X) has the value true. Neither of the values IEEE_NEGATIVE_DENORMAL and IEEE_POSITIVE_DENORMAL shall be returned unless IEEE_SUPPORT_DENORMAL(X) has the value true.

Example. IEEE_CLASS(–1.0) has the value IEEE_NEGATIVE_NORMAL.

15.9.2 IEEE_COPY_SIGN (X, Y)

Description. IEEE copysign function.

Class. Elemental function.

Arguments. The arguments shall be of type real.

Restriction. A program is prohibited from invoking this procedure if IEEE_SUPPORT_DATATYPE(X) or IEEE_SUPPORT_DATATYPE(Y) has the value false.

Result Characteristics. Same as X.

Result Value. The result has the value of X with the sign of Y. This is true even for IEEE special values, such as NaN and Inf (on processors supporting such values).

Examples. The value of IEEE_COPY_SIGN(X,1.0) is ABS(X) even when X is NaN.

15.9.3 IEEE_GET_FLAG (FLAG, FLAG_VALUE)

Description. Get an exception flag.

Class. Elemental subroutine.

Arguments.

FLAG shall be of type TYPE(IEEE_FLAG_TYPE). It is an INTENT(IN) argument and specifies the IEEE flag to be obtained.

FLAG_VALUE shall be of type default logical. It is an INTENT(OUT) argument. If the value of FLAG is IEEE_INVALID, IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, IEEE_UNDERFLOW, or IEEE_INEXACT, the result value is true if the corresponding exception flag is signaling and is false otherwise.

Example. Following CALL IEEE_GET_FLAG(IEEE_OVERFLOW,FLAG_VALUE), FLAG_VALUE is true if the IEEE_OVERFLOW flag is signaling and is false if it is quiet.

15.9.4 IEEE_GET_HALTING_MODE (FLAG, HALTING)

Description. Get halting mode for an exception.

Class. Elemental subroutine.

Arguments.

FLAG shall be of type TYPE(IEEE_FLAG_TYPE). It is an INTENT(IN) argument and specifies the IEEE flag. It shall have one of the values IEEE_INVALID, IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, IEEE_UNDERFLOW, and IEEE_INEXACT.

HALTING shall of type default logical. It is of INTENT(OUT). The value is true if the exception specified by FLAG will cause halting. Otherwise, the value is false.

Example. To store the halting mode for IEEE_OVERFLOW, do a calculation without halting, and restore the halting mode later:

```
USE, INTRINSIC :: IEEE_ARITHMETIC
LOGICAL HALTING
   :
CALL IEEE_GET_HALTING_MODE(IEEE_OVERFLOW,HALTING) ! Store halting mode
CALL IEEE_SET_HALTING_MODE(IEEE_OVERFLOW,.FALSE.) ! No halting
  :  ! calculation without halting
CALL IEEE_SET_HALTING_MODE(IEEE_OVERFLOW,HALTING) ! Restore halting mode
```

Notes: The initial halting mode is processor dependent. Halting is not precise and may occur some time after the exception has occurred.

15.9.5 IEEE_GET_ROUNDING_MODE (ROUND_VALUE)

Description. Get the current IEEE rounding mode.

Class. Subroutine.

Argument. ROUND_VALUE shall be scalar of type TYPE(IEEE_ROUND_TYPE). It is an INTENT(OUT) argument and returns the floating point rounding mode, with value IEEE_NEAREST, IEEE_TO_ZERO, IEEE_UP, or IEEE_DOWN if one of the IEEE modes is in operation and IEEE_OTHER otherwise.

Example. To store the rounding mode, do a calculation with round to nearest, and restore the rounding mode later:

```
USE, INTRINSIC :: IEEE_ARITHMETIC
TYPE(IEEE_ROUND_TYPE) ROUND_VALUE
   :
CALL IEEE_GET_ROUNDING_MODE(ROUND_VALUE) ! Store the rounding mode
CALL IEEE_SET_ROUNDING_MODE(IEEE_NEAREST)
  :  ! calculation with round to nearest
CALL IEEE_SET_ROUNDING_MODE(ROUND_VALUE) ! Restore the rounding mode
```

Note. The result can legally be used only in an IEEE_SET_ROUNDING_MODE invocation.

15.9.6 IEEE_GET_STATUS (STATUS_VALUE)

Description. Get the current values of the set of flags that define the current floating point status, including all the exception flags.

Class. Subroutine.

Arguments. STATUS_VALUE shall be scalar of type TYPE(IEEE_STATUS_TYPE). It is an INTENT(OUT) argument and returns the floating point status.

Example. To store all the exception flags, do a calculation involving exception handling, and restore them later:

```
USE, INTRINSIC :: IEEE_ARITHMETIC
TYPE(IEEE_STATUS_TYPE) STATUS_VALUE
   :
CALL IEEE_GET_STATUS(STATUS_VALUE) ! Get the flags
CALL IEEE_SET_FLAG(IEEE_ALL,.FALSE.) ! Set the flags quiet.
  :  ! calculation involving exception handling
CALL IEEE_SET_STATUS(STATUS_VALUE) ! Restore the flags
```

Note. The result can be used only in an IEEE_SET_STATUS invocation.

15.9.7 IEEE_IS_FINITE (X)

Description. Whether a value is finite.

Class. Elemental function.

Argument. X shall be of type real.

Restriction. A program is prohibited from invoking this procedure if IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Default logical.

Result Value. The result has the value true if the value of X is finite, that is, IEEE_CLASS(X) has one of the values IEEE_NEGATIVE_NORMAL, IEEE_NEGATIVE_DENORMAL, IEEE_NEGATIVE_ZERO, IEEE_POSITIVE_ZERO, IEEE_POSITIVE_DENORMAL, and IEEE_POSITIVE_NORMAL; otherwise, the result has the value false.

Example. IEEE_IS_FINITE(1.0) has the value true.

15.9.8 IEEE_IS_NAN (X)

Description. Whether a value is IEEE Not-a-Number.

Class. Elemental function.

Argument. X shall be of type real.

Restriction. A program is prohibited from invoking this procedure if IEEE_SUPPORT_NAN(X) has the value false.

Result Characteristics. Default logical.

Result Value. The result has the value true if the value of X is an IEEE NaN; otherwise, it has the value false.

Example. IEEE_IS_NAN(SQRT(–1.0)) has the value true if IEEE_SUPPORT_SQRT(1.0) has the value true.

15.9.9 IEEE_IS_NEGATIVE (X)

Description. Whether a value is negative.

Class. Elemental function.

Argument. X shall be of type real.

Restriction. A program is prohibited from invoking this procedure if IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Default logical.

Result Value. The result has the value true if IEEE_CLASS(X) has one of the values IEEE_NEGATIVE_NORMAL, IEEE_NEGATIVE_DENORMAL, IEEE_NEGATIVE_ZERO and IEEE_NEGATIVE_INF; otherwise, the result has the value false.

Example. IEEE_IS_NEGATIVE(0.0)) has the value false.

15.9.10 IEEE_IS_NORMAL (X)

Description. Whether a value is normal, that is, neither an Infinity, a NaN, nor denormalized.

Class. Elemental function.

Argument. X shall be of type real.

Restriction. A program is prohibited from invoking this procedure if IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Default logical.

Result Value. The result has the value true if IEEE_CLASS(X) has one of the values IEEE_NEGATIVE_NORMAL, IEEE_NEGATIVE_ZERO, IEEE_POSITIVE_ZERO and IEEE_POSITIVE_NORMAL; otherwise, the result has the value false.

Example. IEEE_IS_NORMAL(SQRT(–1.0)) has the value false if IEEE_SUPPORT_SQRT(1.0) has the value true.

15.9.11 IEEE_LOGB (X)

Description. Unbiased exponent in the IEEE floating point format.

Class. Elemental function.

Argument. X shall be of type real.

Restriction. A program is prohibited from invoking this procedure if IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Same as X.

Result Value.

Case (i): If X is finite and the value of X is neither zero, infinity, nor NaN, the result has the value of the unbiased exponent of X. Note: this value is equal to EXPONENT(X)–1.

Case (ii): If X==0, the result is –Inf if IEEE_SUPPORT_INF(X) is true and –HUGE(X) otherwise; IEEE_DIVIDE_BY_ZERO signals.

Example. IEEE_LOGB(–1.1) has the value 0.0.

15.9.12 IEEE_NEXT_AFTER (X, Y)

Description. Returns the next representable neighbor of X in the direction toward Y.

Class. Elemental function.

Arguments. The arguments shall be of type real.

Restriction. A program is prohibited from invoking this procedure if IEEE_SUPPORT_DATATYPE(X) or IEEE_SUPPORT_DATATYPE(Y) has the value false.

Result Characteristics. Same as X.

Result Value.

Case (i): If X == Y, the result is X without any exception ever being signaled.

Case (ii): If X /= Y, the result has the value of the next representable neighbor of X in the direction of Y. The neighbors of zero (of either sign) are both nonzero. IEEE_OVERFLOW is signaled when X is finite but IEEE_NEXT_AFTER(X,Y) is infinite; IEEE_UNDERFLOW is signaled when IEEE_NEXT_AFTER(X,Y) is denormalized; in both cases, IEEE_INEXACT signals.

Example. The value of IEEE_NEXT_AFTER(1.0,2.0) is 1.0+EPSILON(X).

15.9.13 IEEE_REM (X, Y)

Description. IEEE REM function.

Class. Elemental function.

Arguments. The arguments shall be of type real.

Restriction. A program is prohibited from invoking this procedure if IEEE_SUPPORT_DATATYPE(X) or IEEE_SUPPORT_DATATYPE(Y) has the value false.

Result Characteristics. Real with the kind type parameter of whichever argument has the greater precision.

Result Value. The result value, regardless of the rounding mode, shall be exactly X – Y*N, where N is the integer nearest to the exact value X/Y; whenever |N – X/Y| = 1/2, N shall be even. If the result value is zero, the sign shall be that of X.

Examples. The value of IEEE_REM(4.0,3.0) is 1.0, the value of IEEE_REM(3.0,2.0) is –1.0 the value of IEEE_REM(5.0,2.0) is 1.0.

15.9.14 IEEE_RINT (X)

Description. Round to an integer value according to the current rounding mode.

Class. Elemental function.

Argument. X shall be of type real.

Restriction. A program is prohibited from invoking this procedure if IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Same as X.

Result Value. The value of the result is the value of X rounded to an integer according to the current rounding mode. If the result has the value zero, the sign is that of X.

Examples. If the current rounding mode is round-to-nearest, the value of IEEE_RINT(1.1) is 1.0. If the current rounding mode is round-up, the value of IEEE_RINT(1.1) is 2.0.

15.9.15 IEEE_SCALB (X, I)

Description. Returns $2^{I}X$.

Class. Elemental function.

Arguments.

X shall be of type real.

I shall be of type integer.

Restriction. A program is prohibited from invoking this procedure if IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Same as X.

Result Value.

Case (i): If $2^{I}X$ is representable as a normal number, the result has this value.

Case (ii): If X is finite and $2^{I}X$ is too large, the IEEE_OVERFLOW exception shall occur. If IEEE_SUPPORT_INF(X) is true, the result value is infinity with the sign of X; otherwise, the result value is SIGN(HUGE(X),X).

Case (iii): If $2^{I}X$ is too small and there is loss of accuracy, the IEEE_UNDERFLOW exception shall occur. The result is the nearest representable number with the sign of X.

Case (iv): If X is infinite, the result is the same as X; no exception signals.

Example. The value of IEEE_SCALB(1.0,2) is 4.0.

15.9.16 IEEE_SELECTED_REAL_KIND ([P, R])

Description. Returns a value of the kind type parameter of a IEEE real data type with decimal precision of at least P digits and a decimal exponent range of at least R. For data objects of such a type, IEEE_SUPPORT_DATATYPE(X) has the value true.

Class. Transformational function.

Arguments. At least one argument shall be present.

P (optional) shall be scalar and of type integer.

R (optional) shall be scalar and of type integer.

Result Characteristics. Default integer scalar.

Result Value. The result has a value equal to a value of the kind type parameter of a IEEE real data type with decimal precision, as returned by the function PRECISION, of at least P digits and a decimal exponent range, as returned by the function RANGE, of at least R, or if no such kind type parameter is available on the processor, the result is –1 if the precision is not available, –2 if the exponent range is not available, and –3 if neither is available. If more than one kind type parameter value meets the criteria, the value returned is the one with the smallest decimal precision, unless there are several such values, in which case the smallest of these kind values is returned.

Example. IEEE_SELECTED_REAL_KIND(6,70) has the value KIND(0.0) on a machine that supports IEEE single precision arithmetic for its default real approximation method.

15.9.17 IEEE_SET_FLAG (FLAG, FLAG_VALUE)

Description. Assign a value to an exception flag.

Class. Elemental subroutine.

Arguments.

FLAG shall be of type TYPE(IEEE_FLAG_TYPE). It is an INTENT(IN) argument. If the value of FLAG is IEEE_INVALID, IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, IEEE_UNDERFLOW, or IEEE_INEXACT, the corresponding exception flag is assigned a value.

FLAG_VALUE shall be of type default logical. It is an INTENT(IN) argument. If it has the value true, the flag is set to be signaling; otherwise, the flag is set to be quiet.

Example. CALL IEEE_SET_FLAG(IEEE_OVERFLOW,.TRUE.) sets the IEEE_OVERFLOW flag to be signaling.

15.9.18 IEEE_SET_HALTING_MODE (FLAG, HALTING)

Description. Controls continuation or halting after an exception.

Class. Elemental subroutine.

Arguments.

FLAG shall be scalar and of type TYPE(IEEE_FLAG_TYPE). It is of INTENT(IN) and shall have one of the values: IEEE_INVALID, IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, IEEE_UNDERFLOW, and IEEE_INEXACT.

HALTING shall be scalar and of type default logical. It is of INTENT(IN). If the value is true, the exception specified by FLAG will cause halting. Otherwise, execution will continue after this exception.

Restriction. A program is prohibited from invoking this procedure if IEEE_SUPPORT_HALTING(FLAG) has the value false.

Example. CALL IEEE_SET_HALTING_MODE(IEEE_DIVIDE_BY_ZERO,.TRUE.) causes halting after a divide_by_zero exception.

Notes: The initial halting mode is processor dependent. Halting is not precise and may occur some time after the exception has occurred.

15.9.19 IEEE_SET_ROUNDING_MODE (ROUND_VALUE)

Description. Set the current IEEE rounding mode.

Class. Subroutine.

Argument. ROUND_VALUE shall be scalar and of type TYPE(IEEE_ROUND_TYPE). It is an INTENT(IN) argument and specifies the mode to be set.

Restriction. A program is prohibited from invoking this procedure unless IEEE_SUPPORT_ROUNDING (ROUND_VALUE,X) is true for any X such that IEEE_SUPPORT_DATATYPE(X) is true.

Example. To store the rounding mode, do a calculation with round to nearest, and restore the rounding mode later:

```
USE, INTRINSIC :: IEEE_ARITHMETIC
TYPE(IEEE_ROUND_TYPE) ROUND_VALUE
   :
CALL IEEE_GET_ROUNDING_MODE(ROUND_VALUE) ! Store the rounding mode
CALL IEEE_SET_ROUNDING_MODE(IEEE_NEAREST)
  :  ! calculation with round to nearest
CALL IEEE_SET_ROUNDING_MODE(ROUND_VALUE) ! Restore the rounding mode
```

15.9.20 IEEE_SET_STATUS (STATUS_VALUE)

Description. Restore the values of the set of flags that define the the floating point status.

Class. Subroutine.

Argument. STATUS_VALUE shall be scalar and of type TYPE(IEEE_STATUS_TYPE). It is an INTENT(IN) argument. Its value shall have been set in a previous invocation of IEEE_GET_STATUS.

Example. To store all the exceptions flags, do a calculation involving exception handling, and restore them later:

```
USE, INTRINSIC :: IEEE_ARITHMETIC
TYPE(IEEE_STATUS_TYPE) STATUS_VALUE
   :
CALL IEEE_GET_STATUS(STATUS_VALUE) ! Store the flags
CALL IEEE_SET_FLAGS(IEEE_ALL,.FALSE.) ! Set them quiet
  :  ! calculation involving exception handling
CALL IEEE_SET_STATUS(STATUS_VALUE) ! Restore the flags
```

Note: Getting and setting may be expensive operations. It is the programmer’s responsibility to do it when necessary to assure correct results.

15.9.21 IEEE_SUPPORT_DATATYPE ([X])

Description. Inquire if the processor supports IEEE arithmetic.

Class. Inquiry function.

Argument. X (optional) shall be scalar and of type real.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the processor supports IEEE arithmetic for all reals (X absent) or for real variables of the same kind type parameter as X; otherwise, it has the value false. Here, support means employing an IEEE data format and performing the binary operations of +, –, and * as in the IEEE standard whenever the operands and result all have normal values.

Example. IEEE_SUPPORT_DATATYPE(1.0) has the value true if default reals are implemented as in the IEEE standard except that underflowed values flush to zero instead of being denormal.

15.9.22 IEEE_SUPPORT_DENORMAL ([X])

Description. Inquire if the processor supports IEEE denormalized numbers.

Class. Inquiry function.

Argument. X (optional) shall of type real. It may be scalar or array valued.

Restriction. A program is prohibited from invoking this procedure with X present if IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the processor supports arithmetic operations and assignments with denormalized numbers (biased exponent $e = 0$ and fraction $f \neq 0$, see subclause 3.2 of the IEEE standard) for all reals (X absent) or for real variables of the same kind type parameter as X; otherwise, it has the value false.

Example. IEEE_SUPPORT_DENORMAL(X) has the value true if the processor supports denormalized numbers for X.

> Notes: The denormalized numbers are not included in the 13.7.1 model for real numbers and all satisfy the inequality ABS(X) < TINY(X). They usually occur as a result of an arithmetic operation whose exact result is less than TINY(X). Such an operation causes IEEE_UNDERFLOW to signal unless the result is exact. IEEE_SUPPORT_DATATYPE(X) is false if the processor never returns a denormalized number as the result of an arithmetic operation.

15.9.23 IEEE_SUPPORT_DIVIDE ([X])

Description. Inquire if the processor supports divide with the accuracy specified by the IEEE standard.

Class. Inquiry function.

Argument. X (optional) shall of type real. It may be scalar or array valued.

Restriction. A program is prohibited from invoking this procedure with X present if IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the processor supports divide with the accuracy specified by the IEEE standard for all reals (X absent) or for real variables of the same kind type parameter as X; otherwise, it has the value false.

Example. IEEE_SUPPORT_DIVIDE(X) has the value true if the processor supports IEEE divide for X.

15.9.24 IEEE_SUPPORT_FLAG (FLAG [, X])

Description. Inquire if the processor supports an exception.

Class. Inquiry function.

Arguments.

FLAG shall be scalar and of type TYPE(IEEE_FLAG_TYPE). Its value shall be one of IEEE_INVALID, IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, IEEE_UNDERFLOW, and IEEE_INEXACT.

X (optional) shall of type real. It may be scalar or array valued.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the processor supports detection of the specified exception for all reals (X absent) or for real variables of the same kind type parameter as X; otherwise, it has the value false.

Example. ALL(IEEE_SUPPORT_FLAG(IEEE_ALL)) has the value true if the processor supports all the exceptions.

15.9.25 IEEE_SUPPORT_HALTING (FLAG)

Description. Inquire if the processor supports the ability to control during program execution whether to abort or continue execution after an exception.

Class. Inquiry function.

Argument. FLAG shall be of type TYPE(IEEE_FLAG_TYPE). It is an INTENT(IN) argument. Its value shall be one of IEEE_INVALID, IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, IEEE_UNDERFLOW, and IEEE_INEXACT.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the processor supports the ability to control during program execution whether to abort or continue execution after the exception specified by FLAG; otherwise, it has the value false. Here, support shall include the ability to change the mode by CALL IEEE_SET_HALTING(FLAG).

Example. IEEE_SUPPORT_HALTING(IEEE_OVERFLOW) has the value true if the processor supports control of halting after an overflow.

15.9.26 IEEE_SUPPORT_INF ([X])

Description. Inquire if processor supports the IEEE infinity facility.

Class. Inquiry function.

Argument. X (optional) shall of type real. It may be scalar or array valued.

Restriction. A program is prohibited from invoking this procedure with X present if IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the processor supports IEEE infinities (positive and negative) for all reals (X absent) or for real variables of the same kind type parameter as X; otherwise, it has the value false.

Example. IEEE_SUPPORT_INF(X) has the value true if the processor supports IEEE infinities for X.

15.9.27 IEEE_SUPPORT_NAN ([X])

Description. Inquire if processor supports the IEEE Not-A-Number facility.

Class. Inquiry function.

Argument. X (optional) shall of type real. It may be scalar or array valued.

Restriction. A program is prohibited from invoking this procedure with X present if IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the processor supports IEEE NaNs for all reals (X absent) of for real variables of the same kind type parameter as X; otherwise, it has the value false.

Example. IEEE_SUPPORT_NAN(X) has the value true if the processor supports IEEE NaNs for X.

15.9.28 IEEE_SUPPORT_ROUNDING (ROUND_VALUE [,X])

Description. Inquire if processor supports a particular rounding mode for IEEE kinds of reals.

Class. Inquiry function.

Arguments.

ROUND_VALUE shall be of type TYPE(IEEE_ROUND_TYPE).

X (optional) shall of type real. It may be scalar or array valued.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the processor supports the rounding mode defined by ROUND_VALUE for all reals (X absent) or for real variables of the same kind type parameter as X; otherwise, it has the value false. Here, support shall include the ability to change the mode by
CALL IEEE_SET_ROUNDING(ROUND_VALUE).

Example. IEEE_SUPPORT_ROUNDING(IEEE_TO_ZERO) has the value true if the processor supports rounding to zero for all reals.

15.9.29 IEEE_SUPPORT_SQRT ([X])

Description. Inquire if the processor implements SQRT in accord with the IEEE standard.

Class. Inquiry function.

Argument. X (optional) shall of type real. It may be scalar or array valued.

Restriction. A program is prohibited from invoking this procedure with X present if IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the processor implements SQRT in accord with the IEEE standard for all reals (X absent) of for real variables of the same kind type parameter as X; otherwise, it has the value false.

Example. IEEE_SUPPORT_SQRT(X) has the value true if the processor implements SQRT(X) in accord with the IEEE standard. In this case, SQRT(–0.0) has the value –0.0.

15.9.30 IEEE_SUPPORT_STANDARD ([X])

Description. Inquire if processor supports all the IEEE facilities defined in this standard.

Class. Inquiry function.

Argument. X (optional) shall of type real. It may be scalar or array valued.

Result Characteristics. Default logical scalar.

Result Value.

Case (i): If X is absent, the result has the value true if the results of all the functions

IEEE_SUPPORT_DATATYPE(), IEEE_SUPPORT_DENORMAL(), IEEE_SUPPORT_DIVIDE(), IEEE_SUPPORT_FLAG(FLAG) for valid FLAG, IEEE_SUPPORT_HALTING(FLAG) for valid FLAG, IEEE_SUPPORT_INF(), IEEE_SUPPORT_NAN(), IEEE_SUPPORT_ROUNDING(ROUND_VALUE) for valid ROUND_VALUE, and IEEE_SUPPORT_SQRT() are all true; otherwise, the result has the value false.

Case (ii): If X is present, the result has the value true if the results of all the functions IEEE_SUPPORT_DATATYPE(X), IEEE_SUPPORT_DENORMAL(X), IEEE_SUPPORT_DIVIDE(X), IEEE_SUPPORT_FLAG(FLAG,X) for valid FLAG, IEEE_SUPPORT_HALTING(FLAG) for valid FLAG, IEEE_SUPPORT_INF(X), IEEE_SUPPORT_NAN(X), IEEE_SUPPORT_ROUNDING(ROUND_VALUE,X) for valid ROUND_VALUE, and IEEE_SUPPORT_SQRT(X) are all true; otherwise, the result has the value false.

Example. IEEE_SUPPORT_STANDARD() has the value false if the processor supports both IEEE and non-IEEE kinds of reals.

15.9.31 IEEE_UNORDERED (X, Y)

Description. IEEE unordered function. True if X or Y is a NaN, and false otherwise.

Class. Elemental function.

Arguments. The arguments shall be of type real.

Restriction. A program is prohibited from invoking this procedure if IEEE_SUPPORT_DATATYPE(X) or IEEE_SUPPORT_DATATYPE(Y) has the value false.

Result Characteristics. Default logical.

Result Value. The result has the value true if X or Y is a NaN or both are NaNs; otherwise, it has the value false.

Example. IEEE_UNORDERED(0.0,SQRT(–1.0)) has the value true if IEEE_SUPPORT_SQRT(1.0) has the value true.

15.9.32 IEEE_VALUE (X, CLASS)

Description. Generate an IEEE value.

Class. Elemental function.

Arguments.

X shall be of type real.

CLASS shall be of type TYPE(IEEE_CLASS_TYPE). The value of is permitted to be:

- IEEE_SIGNALING_NAN or IEEE_QUIET_NAN if IEEE_SUPPORT_NAN(X) has the value true,
- IEEE_NEGATIVE_INF or IEEE_POSITIVE_INF if IEEE_SUPPORT_INF(X) has the value true,
- IEEE_NEGATIVE_DENORMAL or IEEE_POSITIVE_DENORMAL if IEEE_SUPPORT_DENORMAL(X) has the value true,
- IEEE_NEGATIVE_NORMAL, IEEE_NEGATIVE_ZERO, IEEE_POSITIVE_ZERO or IEEE_POSITIVE_NORMAL.

Restriction. A program is prohibited from invoking this procedure if IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Same as X.

Result Value. The result value is an IEEE value as specified by CLASS. Although in most cases the value is processor dependent, the value shall not vary between invocations for any particular X kind type parameter and CLASS value.

Example. IEEE_VALUE(1.0,IEEE_NEGATIVE_INF) has the value –Infinity.

15.10 Examples

Example 1:

```
     MODULE DOT
  ! Module for dot product of two real arrays of rank 1.
  ! The caller must ensure that exceptions do not cause halting.
        USE, INTRINSIC :: IEEE_EXCEPTIONS
        LOGICAL :: MATRIX_ERROR = .FALSE.
        INTERFACE OPERATOR(.dot.)
           MODULE PROCEDURE MULT
        END INTERFACE
     CONTAINS
        REAL FUNCTION MULT(A,B)
           REAL, INTENT(IN) :: A(:),B(:)
           INTEGER I
           LOGICAL OVERFLOW
           IF (SIZE(A)/=SIZE(B)) THEN
              MATRIX_ERROR = .TRUE.
              RETURN
           END IF
  ! The processor ensures that IEEE_OVERFLOW is quiet
           MULT = 0.0
           DO I = 1, SIZE(A)
              MULT = MULT + A(I)*B(I)
           END DO
           CALL IEEE_GET_FLAG(IEEE_OVERFLOW,OVERFLOW)
           IF (OVERFLOW) MATRIX_ERROR = .TRUE.
        END FUNCTION MULT
     END MODULE DOT
```

This module provides the dot product of two real arrays of rank 1. If the sizes of the arrays are different, an immediate return occurs with MATRIX_ERROR true. If overflow occurs during the actual calculation, the IEEE_OVERFLOW flag will signal and MATRIX_ERROR will be true.

Example 2:

```
USE, INTRINSIC :: IEEE_EXCEPTIONS
USE, INTRINSIC :: IEEE_FEATURES, ONLY: IEEE_INVALID_FLAG
! The other exceptions of IEEE_USUAL (IEEE_OVERFLOW and
! IEEE_DIVIDE_BY_ZERO) are always available with IEEE_EXCEPTIONS
TYPE(IEEE_STATUS_TYPE) STATUS_VALUE
LOGICAL, DIMENSION(3) :: FLAG_VALUE
:
CALL IEEE_GET_STATUS(STATUS_VALUE)
CALL IEEE_SET_HALTING_MODE(IEEE_USUAL,.FALSE.) ! Needed in case the
!                  default on the processor is to halt on exceptions
CALL IEEE_SET_FLAG(IEEE_USUAL,.FALSE.)
! First try the "fast" algorithm for inverting a matrix:
MATRIX1 = FAST_INV(MATRIX) ! This must not alter MATRIX.
CALL IEEE_GET_FLAG(IEEE_USUAL,FLAG_VALUE)
IF (ANY(FLAG_VALUE)) THEN
! "Fast" algorithm failed; try "slow" one:
   CALL IEEE_SET_FLAG(IEEE_USUAL,.FALSE.)
   MATRIX1 = SLOW_INV(MATRIX)
   CALL IEEE_GET_FLAG(IEEE_USUAL,FLAG_VALUE)
   IF (ANY(FLAG_VALUE)) THEN
      WRITE (*, *) 'Cannot invert matrix'
      STOP
   END IF
END IF
CALL IEEE_SET_STATUS(STATUS_VALUE)
```

In this example, the function FAST_INV may cause a condition to signal. If it does, another try is made with SLOW_INV. If this still fails, a message is printed and the program stops. Note, also, that it is important to set the flags quiet before the second try. The state of all the flags is stored and restored.

Example 3:

```
USE, INTRINSIC :: IEEE_EXCEPTIONS
LOGICAL FLAG_VALUE
   :
CALL IEEE_SET_HALTING_MODE(IEEE_OVERFLOW,.FALSE.)
! First try a fast algorithm for inverting a matrix.
CALL IEEE_SET_FLAG(IEEE_OVERFLOW,.FALSE.)
DO K = 1, N
   :
   CALL IEEE_GET_FLAG(IEEE_OVERFLOW,FLAG_VALUE)
   IF (FLAG_VALUE) EXIT
END DO
IF (FLAG_VALUE) THEN
! Alternative code which knows that K-1 steps have executed normally.
:
END IF
```

Here the code for matrix inversion is in line and the transfer is made more precise by adding extra tests of the flag.

Example 4:

```
  REAL FUNCTION HYPOT(X, Y)
! In rare circumstances this may lead to the signaling of IEEE_OVERFLOW
! The caller must ensure that exceptions do not cause halting.
   USE, INTRINSIC :: IEEE_ARITHMETIC
   USE, INTRINSIC :: IEEE_FEATURES, ONLY: IEEE_UNDERFLOW_FLAG
! IEEE_OVERFLOW is always available with IEEE_ARITHMETIC
   REAL X, Y
   REAL SCALED_X, SCALED_Y, SCALED_RESULT
   LOGICAL, DIMENSION(2) :: FLAGS
   TYPE(IEEE_FLAG_TYPE), PARAMETER, DIMENSION(2) :: &
          OUT_OF_RANGE = (/ IEEE_OVERFLOW, IEEE_UNDERFLOW /)
   INTRINSIC SQRT, ABS, EXPONENT, MAX, DIGITS, SCALE
! The processor clears the flags on entry
! Try a fast algorithm first
   HYPOT = SQRT( X**2 + Y**2 )
   CALL IEEE_GET_FLAG(OUT_OF_RANGE,FLAGS)
   IF ( ANY(FLAGS) ) THEN
     CALL IEEE_SET_FLAG(OUT_OF_RANGE,.FALSE.)
     IF ( X==0.0 .OR. Y==0.0 ) THEN
       HYPOT = ABS(X) + ABS(Y)
     ELSE IF ( 2*ABS(EXPONENT(X)-EXPONENT(Y)) > DIGITS(X)+1 ) THEN
       HYPOT = MAX( ABS(X), ABS(Y) )!  one of X and Y can be ignored
      ELSE     ! scale so that ABS(X) is near 1
       SCALED_X = SCALE( X, -EXPONENT(X) )
       SCALED_Y = SCALE( Y, -EXPONENT(X) )
       SCALED_RESULT = SQRT( SCALED_X**2 + SCALED_Y**2 )
       HYPOT = SCALE( SCALED_RESULT, EXPONENT(X) ) ! may cause overflow
     END IF
   END IF
! The processor resets any flag that was signaling on entry
END FUNCTION HYPOT
```

An attempt is made to evaluate this function directly in the fastest possible way. This will work almost every time, but if an exception occurs during this fast computation, a safe but slower way evaluates the function. This slower evaluation may involve scaling and unscaling, and in (very rare) extreme cases this unscaling can cause overflow (after all, the true result might overflow if X and Y are both near the overflow limit). If the IEEE_OVERFLOW or IEEE_UNDERFLOW flag is signaling on entry, it is reset on return by the processor, so that earlier exceptions are not lost.

Annex

The differences between this Technical Report its first edition are listed in

`ftp.nag.co.uk/sc22wg5/Nxxxx`

The differences between the semantics and syntax described in this Technical Report and those incorporated in the current draft of the next revision of ISO/IEC 1539-1:1997 are listed in

`ftp.nag.co.uk/sc22wg5/TR155580diffs`

These differences have proved necessary for the reasons explained in the final paragraph of the Foreword.

The lists are also accessible through the www address

`http://www.etrc.ox.ac.uk/SC22WG5`