To: X3J3
From: JOR (Bleikamp)
Subject: Syntax and Edits for Async I/O
Date: Aug 25, 1997

| Issues resolved in this revision:

| - changed READ/WRITE statements ASYNCHRONOUS specifer
| to ASYNCHRONOUS="YES|NO" form, to avoid ambiguity.
|
| - scrapped old text about what variables could not be
| defined, referenced, passed as actual arg, etc., due
| to problems describing which variables the restrictions
| applied to. Replaced this with a new term, the
| "pending I/O storage sequence", a NON-contiguous collection
| of storage units, the actual memory being referenced by the
| I/O list items.
|
| Also added a new term, "pending I/O storage sequence
| AFFECTOR". An affector variable is a
| variable in a scoping unit where at least one
| executable statement
| was executed while the I/O operation is pending, and any
| "use" of a variable in that scoping unit (if it had been
| executed, due to optimizater hoisting the "use" of the
| variable), would have referenced or defined any storage
| unit in the pending I/O storage sequence.

Remaining issues:
| - Finish the edits for affector variables, and prohibiting
| copyin/copyout procedure calls of affector variables while
| the pending I/O is active.

"Notes to the reader" are not notes to be included in the standard.

Text to be included in the standard is either "quoted" or indented.

- -
Note: Non-normative NOTES to be included in the standard are delimited by dashed lines.
- -

Edits to 96-007R1:

In rule R214 (specification-stmt), add:
  or asynchronous-stmt

In rule R426 (component-attr-spec), add:
  or ASYNCHRONOUS

In rule R503 (attr-spec), add:
  or ASYNCHRONOUS

and add a new section (page 57):
 5.1.2.12   ASYNCHRONOUS attribute

| All variables which are pending I/O storage sequence affectors
| (9.4.1.10) shall have the ASYNCHRONOUS attribute in a scoping
| unit in which they appear, if:
| 1) they appear in an executable statement or in a
| specification expression, and
| 2) any executable statement in that scoping unit is executed
| while an asynchronous data transfer operation is pending.

- -
Note: the ASYNCHRONOUS attribute helps a processor determine
which variables are associated with a pending I/O storage
sequence (the actual memory locations to which asynchronous I/O
is being performed). This information is used to disable certain
code motion optimizations.
- -

- - - - - - this following block is no longer part of the edits
 A variable that :
   1) appears in an asynchronous data transfer statement
      input/output list, or

   2) is in a namelist group that is used in an asynchronous
      data transfer statement, and is read or written
      by that data transfer statement, or

   3) is specified in a SIZE= specifier in an asynchronous
      data transfer statement

 and all variables associated, via argument or sequence
 association, with such a variable shall have the ASYNCHRONOUS
 attribute, or shall be a subobject of an object with the
 ASYNCHRONOUS attribute, in a given scoping unit, if :

   1) the variable is referenced, defined, or used as
      an actual argument in a scoping unit other than the
      scoping unit containing the asynchronous
      data transfer statement, and

   2) any executable statement in such a scoping unit
      might be executed while the asynchronous
      data transfer operation is pending.

- - - - - - this preceeding block is no longer part of the edits

If a variable with the ASYNCHRONOUS attribute (implicitly or explicitly) is passed as an actual argument, the corresponding dummy argument shall have the ASYNCHRONOUS attribute.

- -
Note: A data transfer operation is pending when a READ or WRITE statement with an ASYNCHRONOUS= specifier with a value of YES has been executed, but the corresponding wait operation has not yet been executed.
- -

The ASYNCHRONOUS attribute may be specified for any variable, whether or not that variable appears in an asynchronous data transfer statement.

- -
Note: Any variable is permitted to have the asynchronous attribute so users can remove ASYNCHRONOUS= specifiers from data transfer statements, or change the value of ASYNCHRONOUS= specifers to NO, without having to remove the ASYNCHRONOUS attribute from variables used in the asynchronous I/O lists.
- -

- -
Note: The ASYNCHRONOUS attribute is similar to the VOLATILE attribute provided by some processors, and is intended to facilitate traditional code motion optimizations in the presence of asynchronous input/output.
- -

| Add a new section, 5.2.10 (and renumber 5.2.10 and later sections):

| 5.2.10 ASYNCHRONOUS statement

R5xx asynchronous-stmt is ASYNCHRONOUS [::] <object-name-list>

The ASYNCHRONOUS statement specifes the ASYNCHRONOUS attribute for a list of objects.

In rule R905 (OPEN statement connect-spec), add, after PAD= (on its own line)(pg. 140):
or ASYNCHRONOUS = <scalar-default-char-expr>

Add section 9.3.4.11 (page 142/143):

9.3.4.11 ASYNCHRONOUS= specifier in the OPEN statement

The <scalar-default-char-expr> shall evaluate to YES or NO. If an ASYNCHRONOUS= specifier with a value of YES is specified, then READ and WRITE statements for the opened unit may include an ASYNCHRONOUS= specifier with a value of YES in the control information list. An ASYNCHRONOUS= specifier with a value of NO is always permitted in READ and WRITE statements.

The presence of an ASYNCHRONOUS= specifier with a value of YES in a READ or WRITE statement permits, but does not require, a processor to perform the data transfer asynchronously. The WAIT, CLOSE, and file positioning statements may be used to wait for pending asynchronous data transfer operations to complete, and the INQUIRE statement may be used to inquire whether or not pending asynchronous data transfer operations have completed.

Note to the reader: the above rules imply only external unit input / output (not including the "*" unit) may specify an ASYNCHRONOUS= specifier for READs and WRITEs, since internal files and the "*" external unit are not OPENed.

In section 9.3.5 (CLOSE statement), page 143, add the following paragraph and notes after line 5:

Execution of a CLOSE statement causes the processor to wait for all pending asynchronous data transfer operations for the specified unit to complete.

If a CLOSE statement is executed for a unit with pending asynchronous data transfer operations, that CLOSE statement is considered to be the corresponding wait operation for the READ or WRITE statements that initiated those pending asynchronous data transfer operations, and the CLOSE statement is considered to be a data transfer statement for purposes of end of file, end of record, and error processing.

In rule 912 (io-control-spec) (page 144), add:

or ASYNCHRONOUS=<scalar-default-char-initialization-expr>
or ID = <scalar-default-int-variable>

Add the following constraints after the constraint on line 19, page 145:

Constraint: An ASYNCHRONOUS= specifier shall be present if an ID= specifier is present.

Constraint: An ASYNCHRONOUS= specifier shall not be specified with a value of YES if the <io-unit> is an <internal-file-unit> or "*".

Constraint: The <scalar-default-char-initialization-expr> in an ASYNCHRONOUS= specifier shall have the value YES or NO.

Note to the reader: an ID= specifier, typically used in a corresponding WAIT statement, is NOT required in an asynchronous READ or WRITE statement. The user would have to CLOSE the unit (or execute another wait operation) before referencing any storage locations in an input list or namelist, and to NOT define any storage locations referenced by an output list or namelist in an output statement. This allows a knowledgeable user to READ or WRITE massive amounts of data to a file, without ever waiting for completion, as long as they close the file or perform some other wait operation before modifying or referencing any storage locations referenced by an input / output list or namelist.

In section 9.4.1.9 (page 147), first sentence, insert

synchronous

before "nonadvancing", and add the following as the last sentence of that paragraph:

| If an ASYNCHRONOUS= specifier with a value of YES is present in a non-advancing input statement, the storage units specified in the SIZE= specifier become defined with the count of characters transfered when the wait operation corresponding to the non-advancing input statement is executed.

- -
Note: A CLOSE, INQUIRE or a file positioning statement, as well as a WAIT statement, can be a wait operation (9.3.5).
- -

Insert a new section:

9.4.1.10 Asynchronous specifier

An ASYNCHRONOUS= specifier with a value of YES indicates that this data transfer operation can be performed asynchronously. Records read or written by asynchronous data transfer statements are read, written, and processed in the same order as they would have been if the data transfer statement did not contain the ASYNCHRONOUS specifier.

An ASYNCHRONOUS= specifier with a value of YES shall not be present in a READ or WRITE statement unless the OPEN statement for the unit referenced in the READ or WRITE statement contained an ASYNCHRONOUS= specifier with a value of YES.

When a data transfer statement with an ASYNCHRONOUS= specifier with a value of YES is executed, the set of storage units specified by the item list or namelist in the data transfer statememt, as well as the storage units specified by the SIZE= specifier, is called the "pending I/O storage sequence" for this asynchronous data transfer statement.

- -
Note: a "pending I/O storage sequence" is not necessarily a contiguous set of storage units.
- -

In a scoping unit containing a data transfer statement with an ASYNCHRONOUS= specifier with a value of YES, all variables:

1) contained in the input/output list, or
2) contained in a namelist specified in the data transfer statement, or
3) in a SIZE= specifier,

implicitly receive the ASYNCHRONOUS attribute.

A "pending I/O storage sequence affector variable" is a variable appearing in a scoping unit where at least one executable statement is executed while an asynchronous I/O operation is pending, and:

1) any reference or definition of the variable in that instance of that scoping unit references or defines any storage unit in the pending I/O storage sequence for a pending asynchronous data transfer operation, or
2) the variable appears as an actual argument of a procedure which:
   a) is referenced while an asynchronous data transfer operation is pending, or
   b) initiates an asynchronous I/O operation that is still pending when that procedure returns

   and the variable is associated or partially associated with any storage unit in the pending I/O storage sequence for that asynchronous data transfer operation.

| Some pending I/O storage sequence affector variables are
| required to have the ASYNCHRONOUS attribute, as described in
| section 5.1.2.12.

**** need some text about the pending I/O storage sequence
**** ceasing to exist, sort of as described in 14.7.6.

**** need to describe which procedure calls are prohibited
**** when a pending data transfer exists, to avoid problems
**** with copyin/copyout.

**** need to force a dummy arg which is assumed-shape to be
**** associated with the actual arg when the async attribute
**** is present and the actual arg is not a array section with
**** a vector valued subscript.

**** All the edits about READ I/O lists becoming undefined
**** until the wait operation need to be rewritten to
**** take account of the storage units instead.

**** All the edits about WRITE I/O list variables not
**** being redefinable until the wait operation need to be
**** rewritten to take account of the storage units instead.

- - - - - - The following text is no longer part of the edits
When a data transfer statement with an ASYNCHRONOUS=
specifier with a value of YES is executed, the program
shall not execute
any statements that would cause any variable in the
input / output list, namelist, or the variable specified
in a SIZE= specifier to become undefined as described in
14.7.6, until the corresponding wait operation is performed.
When a namelist group name is specified in an input data
transfer statement with an ASYNCHRONOUS= specifier with a
value of YES, any variables in the namelist group that are
not defined by an input data transfer statement are not
subject to the restrictions described in this paragraph.
- - - - - - The preceeding text is no longer part of the edits

- -
Note: This restriction prohibits, among other things,
a RETURN statement, which causes some local stack variables
to become undefined, from being executed when asynchronous
I/O is pending for those local variables.
- -

- - - - - - this following block is no longer part of the edits
When a data transfer statement with an ASYNCHRONOUS=
| specifier with a value of YES is executed, the program shall not execute any statements that would cause the pointer association status of any variable in the input / output list, namelist, or a variable specified in the SIZE= specifier to change, or would cause any such variable to become associated with a different target, as described in 14.6.2, until the corresponding wait operation is performed.
When a namelist group name is specified in a data transfer
| statement, variables in the namelist group not defined by the input data transfer statement are not subject to the restrictions described in this paragraph.
- - - - - - this preceeding block is no longer part of the edits

- -
| Note: The restrictions in this section ensure that certain
| variables (the memory locations associated with those variables actually) referenced in asynchronous data transfer statements must still exist when the corresponding wait operation is performed, including the implicit CLOSE for open units when a program terminates.
- -

- - - - - - this following block is no longer part of the edits
When an input data transfer statement with an
| ASYNCHRONOUS= specifier with a value of YES is executed, the input list or namelist items, and the variable specified in the SIZE= specifier, if any, become undefined until the corresponding wait operation is executed (9.3.5, 9.5).
When a namelist group name is specified in an input data transfer statement, variables in the namelist group not defined by the data transfer statement do not become undefined.

When a data transfer statement with an
| ASYNCHRONOUS= specifier with a value of YES is executed, the item list or namelist items shall not be redefined until the corresponding wait operation is executed (9.3.5, 9.5).
- - - - - - this preceeding block is no longer part of the edits

- - - - - - this following block is no longer part of the edits
After a READ or WRITE statement with an ASYNCHRONOUS=
| specifier with a value of YES is executed, but before the corresponding wait operation is executed, the program shall
| not invoke any procedure where any variable or subobject thereof:

1) in the input list or namelist, or
2) specified in a SIZE= specifier,

is passed as an actual argument, unless :

1) the actual argument is not an array element where the corresponding dummy argument is an array, and
the actual argument passed does not include any storage location defined or referenced by the data transfer statement, or

2) both the actual argument and the corresponding dummy argument have the ASYNCHRONOUS attribute, the actual argument is not an array section with a vector valued subscript, and

   (a) both the actual argument and corresponding dummy argument have the POINTER attribute, or

   (b) the corresponding dummy argument is an assumed shape array, or

   (c) the actual argument is a whole array that is explicit shape, assumed-size, or allocatable, and the corresponding dummy argument is explicit shape or assumed size.

- -
Note: This restriction prevents interactions between actual arguments passed with so-called copyin/copyout semantics and asynchronous I/O.
- -
- - - - - - this preceeding block is no longer part of the edits

Insert a new section 9.4.1.11:

9.4.1.11 ID= specifier

The ID= specifier identifies a variable that is assigned a processor dependent value during the execution of an asynchronous data transfer statement. This value can be used in a WAIT statement to force the processor to wait for a particular data transfer operation to complete.

In section 9.4.4, list item (5), change "namelist" to

namelist, except that if an ASYNCHRONOUS= specifier with a value of YES was also present, the entities specified in the input/output list or namelist become undefined

In section 9.4.4, list item (8), change "defined" to

defined, except that a variable specified in a SIZE= specifier becomes undefined if an ASYNCHRONOUS= specifier with a value of YES was also specified

- - - - - - this following block is no longer part of the edits
In section 9.4.4.4, page 152, before the paragraph that starts "On output ...", insert the following paragraphs:

| If an ASYNCHRONOUS= specifier with a value of YES is specified on an input statement, the list items or namelist variables, and the variable specified in the SIZE= specifier, if any, become undefined until the corresponding wait operation is executed (9.3.5, 9.5). When a namelist group name is specified in an input data transfer statement, variables in the namelist group not defined by the input statement do not become undefined.

- - - - - - this preceeding block is no longer part of the edits

- - - - - - this following block is no longer part of the edits
In section 9.4.4.4, page 152, after the paragraph that starts "On output ...", insert the following paragraphs:

| If an ASYNCHRONOUS= specifier with a value of YES is specified on an output statement, the list items or namelist variables shall not be redefined until the corresponding wait operation is executed (9.3.5, 9.5).

- - - - - - this preceeding block is no longer part of the edits

| If an ASYNCHRONOUS= specifier with a value of YES is specified in a data transfer statement, data transfers may occur during execution of the statement, during execution of the corresponding wait operation, or anywhere in-between. The data transfer operation is considered to be a pending data transfer operation until a corresponding wait operation is performed.

When a data transfer operation is performed asynchronously, any errors that would have caused the ERR= branch on a synchronous READ or WRITE to be taken, and the IOSTAT variable to be defined with a non-zero value, may instead occur during execution of the corresponding wait operation (a WAIT, CLOSE, INQUIRE or file positioning statement) and take the ERR= branch of that wait operation instead. If an ID= specifier is not present in the initiating READ or WRITE statement, the errors may occur during the execution of any subsequent data transfer statement for that same unit, and not just during the corresponding wait operation.

Insert a new section 9.5, and renumber every section thereafter appropriately:

9.5 WAIT statement

Execution of a WAIT statement causes the processor to wait for one of more previously initiated (pending) asynchronous data transfers to complete.

R919 <wait-statement> is WAIT (<wait-spec-list>)

R920 <wait-spec> is [UNIT =]
<external-file-unit>
or IOSTAT =
<scalar-default-int-variable>
or ERR = <label>
or END = <label>
or EOR = <label>
or ID = <scalar-default-int-variable>

Constraint: A <wait-spec-list> shall contain exactly one <external-file-unit> specifier, and may contain at most one of each of the other specifiers.

Constraint: If the optional characters UNIT= are omitted from the unit specifier, the unit specifier shall be the first item in the <wait-spec-list>.

(note to Richard Maine: insert other appropriate constraints, similar to the position-spec constraints, and one for the END=label branch target)

The IOSTAT=, ERR=, and END= specifiers are described in x, x, and x respectively.

If an ID= specifier is not present, the processor waits for all pending data transfers on the specified unit to complete, if any. If an ID= specifier is present, the processor waits for the corresponding READ or WRITE operation to complete. The corresponding READ or WRITE operation is that READ or WRITE that returned the same value for the ID= specifier for the specified unit. The value specified for the ID= specifier shall be a value returned by a READ or WRITE statement for the specified unit, for which a corresponding wait operation has not been executed.

The data transfer operation specified in the corresponding READ or WRITE statement may happen when the WAIT statement is executed, when the corresponding READ or WRITE statement was previously executed, or anytime in-between. The WAIT statement is considered to be a data transfer statement for purposes of end of file, end of record, and error processing.

- -
Note: The CLOSE , INQUIRE, and file positioning statements, as well as the WAIT statement, can be a "wait" operation.
- -

- -
Note: If an asynchronous READ attempts to read beyond the end of a file, then the end of file condition may occur either during execution of the READ statement or during execution of the corresponding wait operation. If the end of file condition occurs during the wait operation, and there is not an END= or IOSTAT= specifier in the statement that is the corresponding wait operation, then program execution terminates. Error conditions are handled in a similar manner.
- -

and renumber all subsequent rules.

In the old section 9.5 (File Positioning statements), add the following after the last sentence in that section:

Execution of a file positioning statement causes the processor to wait for all pending data transfer operations for the specified unit to complete.

If a file positioning statement is executed for a unit with pending data transfer operations, that statement is considered to be the corresponding wait operation for the READ or WRITE statements that initiated the pending data transfers, and is also considered to be a data transfer statement for purposes of end of file, error, and end of record processing.

In section 9.6.1, add the following to rule 924:
or ID = <scalar-default-int-variable>
or PENDING = <scalar-default-logical-variable>
or ASYNCHRONOUS = <scalar-default-char-variable>

and add these constraints around line 40 on page 156:
Constraint: The ID= and PENDING= specifiers shall not appear in an INQUIRE statement if the FILE = specifier is present.

Constraint: If an ID= specifier is present, a PENDING= specifier shall also be present.

On page 159, add section 9.6.1.23

9.6.1.23 ID= and PENDING= specifiers in the INQUIRE statement

If an ID= specifier is not present in an INQUIRE statement, the variable specified in the PENDING= specifier is assigned the value true if there are any pending asynchronous data transfers for the specified unit that have not completed. If an ID= specifier is present, the variable specified in the PENDING= specifier is assigned the value true if the data transfer identified by the ID= specifier for the specified unit has not yet completed. In all other cases, the variable specified in the PENDING= specifier is set to false.

When the variable specified in the PENDING= specifier is set to false, then any pending data transfer operations for this unit are considered to have completed, and this INQUIRE is the corresponding wait operation for the corresponding READ or WRITE statements. When an ID= specifier is present, the corresponding operation is the READ or WRITE statement identified by the unit and ID= specifier value. When an ID= specifier was not present, then this INQUIRE statement is the corresponding wait operation for all pending data transfer operations for the specified unit. When an INQUIRE statement is considered to be a wait operation, it is also considered to be a data transfer statement for purposes of end of file, end of record, and error processing.

On page 159, add section 9.6.1.24

9.6.1.24 ASYNCHRONOUS= specifier in the INQUIRE statement

The <scalar-default-char-variable> in the ASYNCHRONOUS= specifier is assigned the value "YES" if the file is connected and was opened with an ASYNCHRONOUS= specifier with a value of "YES"; otherwise, it is assigned the value "NO".

In section 9.6.1.14 (page 158), add the following sentence as the last sentence of the paragraph.

If there are pending data transfer operations for the specified unit, the value assigned to the variable specified in a NEXTREC= specifier is computed as if all the pending data transfers had already completed.

Note to the reader: the POSITION= specifier does not appear to need any edits.

Note to the reader. In section 14, we discuss events causing definition and undefinition of variables. In item (3) of 14.7.5, we discuss when input causes an item to be defined, in terms of when the data is transferred, so no edit is needed in (3). Note that the second part of (3) applies to internal units, which cannot be read from or written to asynchronously.

In section 12.3.1.1, add this item under the list (2)

(f) A dummy argument that has the ASYNCHRONOUS attribute, or

and delete the trailing " or" from item (e) in that list.

In section 14.7.5, item (5), change "an input/output statement" to "an input/output statement without an ASYNCHRONOUS= specifier, or an input/output statement with an ASYNCHRONOUS= specifier with a value of NO".

In section 14.7.5, item (8), change "statement" to "statement without an ASYNCHRONOUS specifier or with an ASYNCHRONOUS= specifier with a value of NO".

- - - - - - this following block is no longer part of the edits
In section 14.7.5, insert this new item (9), and renumber the remaining items:

(9) Execution of a READ statement containing both an ASYNCHRONOUS= specifier with a value of YES and a SIZE= specifier may cause the variable specified in the SIZE= specifier to become defined, or the corresponding wait operation may cause that variable to become defined. Either the READ statement or the corresponding wait operation will cause that variable to become defined.

In section 14.7.6, item (4), change "input/output statement" to "input/output statement or its corresponding wait operation".

In section 14.7.6, item (5), change "input/output statement" to "input/output statement or its corresponding wait operation".

In section 14.7.6, item (7), change "input statement" to "input statement or its corresponding wait operation".
- - - - - - this preceeding block is no longer part of the edits

In section 14.7.6, add a new item (16) (the editor may relocate to another part of the list if desired):

Execution of a READ statement with an ASYNCHRONOUS= specifier with a value of YES causes all variables in the item list or namelist, and the variable specified in the SIZE= specifier, if any, to become undefined. Variables in a namelist group specified in such a READ statement that are not defined by the data transfer statement do not become undefined.

Rationale for Asynchronous I/O: may be inserted in the appropriate annex if desired.

Rather than limit support for asynchronous I/O to what has been traditionally provided by facilities such as BUFFERIN-BUFFEROUT, this standard builds upon existing Fortran syntax. This permits alternative approaches for implementing asynchronous I/O, and simplifys the task of adapting existing standard conforming programs to utilize asynchronous I/O.

Not all processors will actually perform I/O asynchronously, nor will every processor that does, be able to handle data transfer statements with complicated I/O item lists in an asynchronous manner. Such processors can still be standard conforming. Hopefully, the documentation for each Fortran processor will describe when, if ever, I/O will be performed asynchronously.

Conceptual Model

This proposal allows for at least two different conceptual models for asynchronous I/O.

Model 1: the processor will perform asynchronous I/O when the item list is simple (perhaps one contiguous named array) and the I/O is unformatted (possibly MAGTAPE). The implementation cost is reduced, and this is the scenario most likely to be beneficial on traditional "big-iron" machines.

Model 2: The processor is free to do any of the following:
  on output, create a buffer inside the I/O library, completely formatted, and then start an async write of the buffer, and immediately return to the next statement in the program. The processor is free to wait for previously issued WRITEs, or not.
OR
  pass off the I/O list addresses to another processor/process, that will process the list items independently of the processor which executes the user's code.
  The addresses of the list items must be computed before the asynchronous READ/WRITE statement completes.
  There is still an ordering requirement on list item processing, to handle things like READ (...) N,(a(i),i=1,N).

One source of confusion is the role of the ID= values and wait operations. The standard allows a user to issue a large number of asynchronous I/O requests, without waiting for any of them to complete, and then wait for any or all of them. It may be impossible, and undesirable to keep track of each of these I/O requests individually.

The proposed support does not require all requests to be tracked by the runtime library. When the user does NOT specify an ID= specifier on a READ or WRITE, the runtime is free to forget about this particular request once it has successfully completed. If it gets an ERR or END condition, the processor is free to report this during any I/O operation to that unit.

When an ID= specifier is present, the processors runtime I/O library is required to keep track of any END or ERR conditions for that specific I/O request.
However, if the I/O request succeeds without any exceptional conditions occuring, then the runtime can forget that ID= value if it wishes. Typically, I expect a runtime to only keep track of the last request made, or perhaps a very few. Then, when a user WAITs for a particular request, either the library knows about it (and does the right thing w.r.t. error handling, etc.), or will assume it is one of those requests that successfully completed and was forgotten about (and will just return without signaling any end/err conditions). It is incumbent on the user to pass valid ID= values. There is no requirement on the processor to detect invalid ID= values.

There is of course, a processor dependent limit on how many outstanding I/O requests which generate an END or ERROR condition can be handled before the processor runs out of memory to keep track of such stuff.

The restrictions on the SIZE= variables are designed to allow the processor to update such variables at any time (after the request has been processed, but before the WAIT operation), and then forget about them. That's why there is no SIZE= specifier allowed in the various WAIT operations. Only exceptional conditions (errors or EOFs) are expected to be tracked by individual request by the runtime, and then only if an ID= specifier was present.

The END= and EOR= specifiers have not been added to all statements which can be WAIT operations. Instead, the IOSTAT variable will have to be queried after a WAIT operation to handle this situation. This choice was made because we expect the WAIT statement to be the usual method of waiting for I/O to complete (and WAIT does support the END= and EOR= specifiers).
This particular choice is philosophical, and was not based on significant technical difficulties.

Note that the requirement to set the IOSTAT variable correctly requires an implementation to remember which I/O requests got an EOR condition, so that a subsequent wait operation will return the correct IOSTAT value. This means there is a processor defined limit on the number of outstanding non-advancing I/O requests which got an EOR condition (constrained by available memory to keep track of this info, similar to END/ERR conditions).