Chapter 13

File Input/Output

This chapter describes the input/output statements (R13-1), including the various forms of READ statements, WRITE statements, and PRINT statements.

A (file) *input-output-stmt* is one of: R13-1

- *read-stmt* ! input
- *write-stmt* ! output
- *print-stmt* ! output

There are auxiliary I/O statements that describe or inquire about external files, and FORMAT statements that control editing of information during a file operation. Auxiliary I/O statements are described in Chapter 14, and the FORMAT statement is described in Chapter 15.

13.1 External Files

An *external file* is a sequence of records that exists on a device or medium external to an executable program. An external file may have a name, in which case it is called a *named file*. The name of a file is a character string, and must be a valid UNIX filename. All I/O statements may refer to files that exist. An INQUIRE, OPEN, CLOSE, WRITE, PRINT, REWIND, or ENDFILE statement may also refer to a file that does not exist.

Every external file has a set of allowed *access methods* (either *sequential* or *direct*) by which the file may be accessed. Some files may support both access methods, while other files may support only a single access method. For example, the run-time system may allow only sequential access to a file on magnetic tape. The method used to access a file is determined when the file is *connected* to a unit during program execution. A file may be *preconnected* (connected before program execution begins), or it may be connected explicitly using an OPEN statement.

Copyright © 1989–1994 Thinking Machines Corporation

Every external file associated with a program has a set of allowed *forms* (either *formatted* or *unformatted*) by which it may be accessed. The records of an external file are either all formatted or all unformatted, except that the last record of an external file may be an end-file record. Section 13.3 describes the forms of formatted and unformatted records.

13.1.1 Sequential Access to External Files

An external file that is connected for *sequential access* may be regarded as a sequence of zero or more formatted or unformatted records, effectively terminated by an endfile record to mark the end of the file. An endfile record may be written explicitly by the `ENDFILE` statement to a file connected for sequential access. An endfile record is written implicitly when the last operation on a file open for sequential access is an output statement (other than an `ENDFILE` statement) and (1) a `REWIND` or `BACKSPACE` statement references the unit, or (2) the unit (file) is closed, either explicitly by a `CLOSE` statement or implicitly when a program terminates normally (not by an error condition).

An endfile record may occur only as the last record of a file. An endfile record does not have a length.

An *end-of-file condition* occurs if an endfile record is encountered during the reading of a file connected for sequential access. An end-of-file condition may be detected by a sequential `READ` statement that includes the `END=` or `IOSTAT=` specifiers in its control list.

When connected for *sequential access*, an external file has the following properties:

- The order of the records is the order in which they were written if the file can only be accessed sequentially. If the file can also be accessed in direct access mode, the order of the records is in order of increasing record number, beginning with record number one. A record that has not been written must not be read.
- The records of the file are either all formatted or all unformatted, except that the last record of the file may be an endfile record. Unless the previous operation on the file was an output statement, the last record (if any) of the file must be an end-file record.
- The records of the file must not be read or written by a direct access I/O statement.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

13.1.2 Direct Access to External Files

An external file that is connected for *direct access* may be regarded as a set of zero or more formatted or unformatted records that may be read or written in arbitrary order. A file connected for direct access does not have an endfile record.

When connected for direct access, an external file has the following properties:

- Each record of the file is identified by an integer value called the *record number,* with a value of one or greater. The record number is specified when the record is written (using the `REC=` specifier). Once established, the record number of a record can never be changed. Note that a record may not be deleted, but it may be rewritten. The order of the records is the order of their record numbers.
- Records need not be read or written in the order of their record numbers. Any record may be written into the file while it is connected to a unit. For example, it is permissible to write record 3 even though records 1 and 2 have not been written. Any record may be read from the file while it is connected to a unit, provided that the record has been written since the file was created.
- All records of the file have the same length.
- The records of a file are either all formatted or all unformatted. If the file can also be opened for sequential access, then its endfile record is not considered to be part of the file while it is connected for direct access.
- The records of the file may not be read or written using list-directed formatting.
- Reading and writing records is accomplished only by direct access I/O statements.

The `OPEN` statement (Section 14.2) specifies the size and form of the records in a direct access file (using the `RECL=` specifier).

For a file connected for direct access, the record number is increased by one as each succeeding record is read or written. An attempt to read a record of a file connected for direct access that has not previously been written causes all entities specified by the input list to become undefined.

13.1.3 Position of External Files

An external file that is connected has an attribute called its *position*. The *initial point* of a file is the position just before the first record. A new file is positioned at its initial point when it is opened. The position of an existing file is determined by the `POSITION=` speci-

Copyright © 1989–1994 Thinking Machines Corporation

fier in the OPEN statement (Section 14.2.8). The *terminal point* is the position just after the last record (including just after the endfile record, if the file is a sequential file).

If a file is positioned at the beginning of a record or within a record, that record is the *current record* of the file; otherwise, the file has no current record.

Let n be the number of records in a file. If $1 < i \leq n$ and the file is positioned within the ith record or between the $(i-1)$th record and the ith record, the $(i-1)$th is the *preceding record*. If $n \geq 1$ and the file is positioned at its terminal point, the preceding record is the nth and last record. If $n = 0$ or if a file is positioned at its initial point or within the first record, there is no preceding record.

If $1 \leq i < n$ and a file is positioned within the ith record or between the ith and $(i + 1)$th record, the $(i + 1)$th record is the *next* record. If $n \geq 1$ and the file is positioned at its terminal point or within the nth record, there is no next record.

The positioning of the file *prior* to data transfer depends on the method of file access (sequential or direct). **NOTE:** For information about the position of the file after the OPEN statement, see Section 14.2.8.

> For sequential access on input, the file is positioned at the beginning of the next record. This record becomes the current record. On output, a new record is created and becomes the last record of the file.
>
> For direct access, the file is positioned at the beginning of the record specified by the REC= specifier. This record becomes the current record.
>
> If the file contains an endfile record, the file must not be positioned after the endfile record prior to data transfer.

If an end-of-file condition exists as a result of reading an endfile record and no error condition exists, the file is positioned *after* the endfile record. If an end-of-file condition exists as a result of reading the last record of the file (not an endfile record), the file is positioned after the last record of the file.

If no error condition or end-of-file condition exists, the file is positioned after the last record read or written and that record becomes the preceding record. A record written to a file connected for sequential access becomes the last record of the file.

If the file is positioned after the endfile record, execution of a READ, WRITE, or PRINT statement is prohibited. However, a REWIND or BACKSPACE statement may be used to reposition the file.

If an error condition exists after a data transfer operation, the position of the file is indeterminate.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

13.2 Internal Files

Internal files provide a means of using formatted and list-directed I/O statements to convert program variables from their internal binary representation to ASCII format, or from ASCII format to an internal binary representation. An *internal file* has the following properties:

- The file is a character variable (a scalar or an array). (A CM array may not be used as an internal file, since character arrays are not allocated in CM memory.)
- If the file is a *scalar* character variable, it consists of a *single* record whose length is the same as the length of the scalar character variable.

  If the file is an *array* character variable, it is treated as a sequence of records. Each array element is a record of the file, and each record has the same length. The ordering of the records of the file is the same as the ordering of the array elements in the array.
- A record of the internal file becomes defined by writing the record. If the number of characters written is fewer than the length of the record, the remaining portion of the record is filled with blanks. If the number of characters to be written is greater than the length of the record, then only enough characters to fill the record are written and the remaining characters are truncated.
- A record should be read only if the value of the record is defined.
- A record of an internal file may become defined (or undefined) by means other than an output statement. For example, the character variable may become defined by a character assignment statement.
- An internal file is always positioned at the beginning of the first record prior to data transfer. This record becomes the current record.
- On input, leading blanks are skipped. (They are treated as though the record is read with a format specification that has an initial `BN` edit descriptor, as described in Chapter 15.)
- There are no endfile records in internal files, but an end-of-file condition can be detected using an `ERR=` or an `IOSTAT=` specifier. An end-of-file condition occurs if an attempt is made to read a record beyond the end of an internal file.

Reading and writing of records of an internal file must be accomplished using sequential, formatted (including list-directed) I/O statements, as described in the following sections. The language prohibits the use of internal files in I/O statements other than these forms of `READ`, `WRITE`, and `PRINT` statements.

Copyright © 1989–1994 Thinking Machines Corporation

13.3 Record Formats

A file comprises a series of *records*, each of which is a sequence of binary values or a sequence of characters. A line entered from a terminal can be considered to be a record. A record does not necessarily correspond to a physical entity. The three kinds of records — *formatted*, *unformatted*, and *endfile* — are discussed in the following sections.

13.3.1 Formatted Records

A formatted record consists of a sequence of ASCII characters, possibly grouped into a series of *fields* of different lengths. The length of a formatted record is measured in characters and depends primarily on the number of characters put into the record when it is written, but it may depend as well on the external medium. A formatted record may have a length of zero.

Formatted records may be read or written only by formatted I/O statements. Formatted records may be prepared by some means other than CM Fortran; for example, by some manual input device.

13.3.1.1 Records Formatted for List-Directed I/O

Formatted records written using list-directed `WRITE` or `PRINT` statements have a format described in Chapter 15. Records written using list-directed output statements generally are not suitable for reading by list-directed input statements.

13.3.2 Unformatted Records

An unformatted record consists of a sequence of binary values. The length of an unformatted record is measured in bytes and depends upon the output list used when it is written. The length of an unformatted record may be zero. Unformatted records may be read or written only by unformatted data transfer statements.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

13.3.3 Endfile Records

An *endfile record* is written by an ENDFILE statement when a file is open for sequential access. An endfile record may occur only as the last record of a file. An endfile record does not have a length.

If a file may be accessed using either the sequential mode or the direct access mode, an endfile record written during sequential I/O operations to the file is "invisible" if the file is later opened for direct access.

13.4 Form of the File I/O Statements

The READ statement (R13-2) is an *input statement* that moves data from an external file to internal storage or from an internal file to internal storage. The WRITE statement (R13-3) and the PRINT statement (R13-4) are *output statements* that move data from internal storage to an external file or from internal storage to an internal file.

A *read-stmt* is one of: R13-2

- READ (*io-control-spec(s)*) *[input-item(s)]*
- READ *format-identifier [, input-item(s)]*

A *write-stmt* is one of: R13-3

- WRITE (*io-control-spec(s)*) *[output-item(s)]*
- WRITE *format-identifier [, output-item(s)]*

A *print-stmt* is: R13-4

PRINT *format-identifier [, output-item(s)]*

All forms of file I/O statements may include an input/output item list (I/O list) of items to be read or written. The most general form of READ and WRITE statements include a parenthesized *I/O control specification list* that supplies additional information about the type of operation. A simple form of the READ and WRITE statements may be used when only a *format-identifier* is needed to specify the formatting to be performed. The PRINT statement is an abbreviated output statement that can replace a WRITE statement for some sequential I/O operations. Input/output control specifiers are described in Section 13.4.1, and I/O item lists are described in Section 13.4.2.

NOTE: INTEGER*8 values cannot be used in READ, WRITE, and PRINT statements as anything other than the I/O list items (the data to be transferred). This type cannot be used for unit numbers, record numbers, status variables, and so on.

Copyright © 1989–1994 Thinking Machines Corporation

13.4.1 Input/Output Control Specification List

A READ or WRITE statement may contain an *I/O control specification list* (R13-5) that provides information about the operation to be performed by the statement.

An *io-control-spec* is one of: R13-5

- *[* UNIT = *]* *io-unit-identifier* ! required
- *[* FMT = *]* *format-identifier*
- *[* NML = *]* *namelist-name*
- REC = scalar-integer-*expr*
- END = *label*
- ERR = *label*
- IOSTAT = scalar-integer-*variable*

- An *io-control-spec(s)* list must contain exactly one UNIT= specifier and may contain at most one of each of the other specifiers.
- If the optional phrase "UNIT =" is omitted from a UNIT= specifier, then the *io-unit-identifier* must be the first specifier in the *io-control-spec(s)* list.
- If the optional phrase "FMT =" is omitted from a FMT= specifier, the *format-identifier* must be the second item in the *io-control-spec(s)* list and the first item must be a UNIT= specifier without the phrase "UNIT =".
- If the optional phrase "NML =" is omitted from a NML= specifier, the *namelist-name* must be the second item in the *io-control-spec(s)* list and the first item must be a UNIT= specifier without the phrase "UNIT =".
- The *label* used in an ERR= or END= specifier must be the statement label of a branch target statement that appears in the same program unit as the file I/O statement.
- The REC= specifier may appear only in direct access file I/O statements, as indicated in the restricted forms of *io-control-spec* defined in Section 13.7.
- The END= specifier may appear only in a sequential READ statement, as indicated in the restricted forms of *io-control-spec* defined in Section 13.6.

An expression in an *io-control-spec* must not reference a function that causes another I/O statement to be executed.

The UNIT= specifier indicates the source or destination of the data to be transferred (the external or internal file that is to be accessed during the data transfer). A FMT= specifier indicates how data is to be edited during formatted data transfer. A REC= specifier designates the record to be read or written in direct access mode. The END= specifier indicates a branch target statement to receive control in case an end of file condition is detected

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

during a sequential read operation. The ERR= specifier indicates a branch target statement to receive control in case an error condition is detected during an I/O operation. The IOSTAT= specifier names a variable to receive a status code indicating the status of the operation.

If the file I/O statement contains a *format-identifier*, the statement is a *formatted* I/O statement; otherwise, it is an *unformatted* I/O statement.

An *io-control-spec* that contains a FMT= specifier of * establishes list-directed formatting (described in Sections 13.5.5 and 13.5.6 for external files, and in Sections 13.6.3 and 13.6.4 for internal files). If the I/O control list does not contain a FMT= specifier of *, the format specification identified by the FMT= specifier is established.

13.4.1.1 UNIT= Specifier

The UNIT= I/O control specifier is a parameter that identifies the unit to be accessed by an I/O statement.

A (UNIT) *io-control-spec* is: R13-5
- *[* UNIT = *]* *unit-number*
- *[* UNIT = *]* *
- *[* UNIT = *]* *internal-file-spec*

A *unit-number* is: R13-6
scalar-integer-*expr*

An *internal-file-spec* is: R13-7
char-variable

The UNIT= specifier provides a means of referring to a specific file or a connection. A unit is either an external unit or an internal unit. An *external unit* refers to an external file and is specified by a *unit-number* or an asterisk (*). An *internal-file-spec* is the name of a character variable that may be read or written sequentially using a READ or a WRITE statement, as described in Section 13.6. The *io-unit-identifier* must be a *unit-number* if it appears in a file positioning statement, a file connection statement, or a file inquiry statement (these statements are described in Chapter 14).

Copyright © 1989–1994 Thinking Machines Corporation

The external unit identified by a scalar-integer-*expr* is the same in all subprograms and the main program comprising an executable program. In the example

```
SUBROUTINE A
READ ( 5+1 ) X
...
END

SUBROUTINE B
N = 6
REWIND N
...
END
```

the value 6 used in both program units identifies the same external unit.

A scalar integer expression representing a unit number must be zero or positive.

An *io-unit-identifier* that is an asterisk identifies a particular external unit that is preconnected for formatted sequential access. When used in `READ` statements, an asterisk identifies the standard input file; in `WRITE` statements, an asterisk identifies the standard output file.

13.4.1.2 FMT= Specifier

The `FMT=` I/O control specifier identifies a format to be used during a formatted data transfer.

A (FMT) *io-control-spec* is: R13-8

- *[* `FMT =` *]* `*`
- *[* `FMT =` *]* *label*
- *[* `FMT =` *]* *char-expr*
- *[* `FMT =` *]* scalar-integer-*variable*

♦ A `FMT=` specifier that is a *label* must be the label of a `FORMAT` statement that appears in the same program unit as the file I/O statement in which it is used.

The kind of `FMT=` specifier supplied in an I/O control list indicates where format control information should be obtained during the data transfer.

- A `*` specifies that list-directed I/O is to be performed. (List-directed I/O is described in Sections 13.5.5, 13.5.6, 13.6.3, 13.6.4.)
- A *label* specifies that formatted I/O is to be performed under control of the `FORMAT` statement specified. (`FORMAT` statements are described in Chapter 15.)

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

- A scalar integer variable indicates that formatted I/O is to be performed under control of a FORMAT statement. A variable must have been assigned the statement label of a FORMAT statement using the ASSIGN statement.
- A character expression must evaluate to a character value that is a valid format specification. The character value must begin with a left parenthesis and end with a right parenthesis, and may contain one or more edit descriptors. Only the edit descriptors described in Chapter 15 on the FORMAT statement can be used between the parentheses.

  If the FMT= specifier is a character array, its individual elements are effectively concatenated in array element order, and the resulting character string controls the formatting.

  An apostrophe in a constant string that is enclosed in apostrophes must be represented by two consecutive apostrophes.

  A character expression that forms a FMT= specifier must not reference a function that can cause another I/O statement to be executed.

An example in which the FMT= specifier is a character expression is

```
READ ( 6, FMT = "(" // FMTSTR // ")" ) X, Y, Z
```

where FMTSTR is a character variable containing one or more valid edit descriptors.

13.4.1.3 NML= Specifier

The NML= I/O control specifier supplies a *namelist-name*.

An (NML) *io-control-spec* is: R13-5
NML = *namelist-name*

The *namelist-name* identifies the set of variables whose values are to be read or written. A *namelist-name* is defined by a NAMELIST statement as described in Section 5.7. Namelist editing is described in detail in Section 15.7.

The input and output records begin with an ampersand character (&) followed immediately by the *namelist-name* and then a list of name-value subsequences terminated by a slash. (An alternative form substitutes a dollar sign ($) for the ampersand and terminates the list of name-value subsequences with $END instead of a slash.) A name-value subsequence consists of the name of one of the variables defined to be in the namelist group by a NAMELIST statement, an equals sign (=), a value, and a field separator. The variable specified may be an array element, array section, or a substring. Input values must be in a format appropriate to the type of the variable. These formats are the same as those used for list-

Copyright © 1989–1994 Thinking Machines Corporation

directed input and are described in detail in Section 15.6.1. Output values are written in the standard formats described in detail in Section 15.6.2. The field separators are the same as those used in list-directed I/O described in Section 15.6. The separators are blanks, tabs, commas, or slashes.

An example of the use of a namelist input statement is

```
INTEGER I
REAL X(8)
COMPLEX Z
LOGICAL G

NAMELIST /NLIST/ I,X,Z,G
READ (*, NML=NLIST)
.
.
.
END
```

If the input data record is

```
&NLIST I=12345, X(1)=12345, X(3:4)=2*1.5, Z=(123,0)/
```

the results stored by the READ statement are

```
I              12345
X(1)           12345.0
X(2)           unchanged
X(3)           1.5
X(4)           1.5
X(5) to X(8)   unchanged
Z              (123.0,0.0)
G              unchanged
```

Note that variables in the namelist that are not specified in the input data record are not changed.

13.4.1.4 REC= Specifier

The REC= I/O control specifier indicates the number of a record to be read or written by a file I/O statement.

A (REC) *io-control-spec* is: R13-5
    REC = scalar-integer-*expr*

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

The presence of a REC= specifier in an I/O statement defines that statement as a *direct access* operation. Such a statement must specify a unit that is connected to a file for direct access.

13.4.1.5 END= Specifier

The END= I/O control specifier supplies the label of an executable statement to receive control when an end-of-file condition is detected during a sequential READ operation.

An (END) *io-control-spec* is: R13-5
    END = *label*

- The *label* must be the label of an executable branch target statement in the same program unit.

The control information list of a WRITE statement may not contain an END= specifier.

If an END= specifier detects an end-of-file condition (and no error condition is detected), then:

If an I/O statement contains an END= specifier and the program detects an end-of-file condition during execution of the statement (and no error condition is detected), the following actions occur:

- Execution of the READ statement terminates,
- If the file specified in the input statement is an external file, it is positioned after the endfile record,
- If the input statement also contains an IOSTAT= specifier, the variable specified becomes defined with a −1, and
- Execution continues with the statement specified in the END= specifier.

13.4.1.6 ERR= Specifier

The ERR= I/O control specifier supplies the label of an executable statement to receive control if an error is detected.

An (ERR) *io-control-spec* is: R13-5
    ERR = *label*

Copyright © 1989–1994 Thinking Machines Corporation

- The *label* must be the label of an executable branch target statement in the same program unit.

If an I/O statement contains an ERR= specifier and the program detects an error condition during execution of the statement, the following actions occur:

- execution of the I/O statement terminates
- the position of the file specified in the I/O statement becomes indeterminate
- the variable specified in an IOSTAT= specifier, if any, becomes defined with a positive integer value, and
- execution continues with the statement specified in the ERR= specifier.

On input, all list items are undefined. Note that for list-directed input, some elements of the input list may not have had their value changed due to the presence of null values in the input file.

13.4.1.7 IOSTAT= Specifier

The IOSTAT= I/O control specifier designates a variable to be assigned the status code returned by an I/O statement.

An (IOSTAT) *io-control-spec* is: R13-5
IOSTAT = scalar-integer-*variable*

Execution of an I/O statement containing an IOSTAT= specifier causes the specified variable to become defined with one of the following values:

0	If the operation is successful and end-of-file is not detected.
−1	If end-of-file is reached and no error occurs. (Note that an end-of-file condition may occur only during sequential input.)
n > 0	If an error occurs. See Section 13.8 for a list of the I/O status codes.

A variable referenced in an IOSTAT= specifier of a file I/O statement must not be associated with any item in the I/O item list, nor with an index variable of an I/O implied DO control in the file I/O statement input or output list.

If the variable referenced in an IOSTAT= specifier is an array element reference, its subscript values must not be affected by the data transfer, nor by the I/O implied DO processing, nor by the definition or evaluation of any other specifier in the I/O item list.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

The use of the IOSTAT= specifier is illustrated by the following example:

```
...
READ ( FMT = "(E8.3)", UNIT=3, IOSTAT = IOSS ) X
IF ( IOSS < 0 ) THEN
  CALL EOF_PROCESSING
ELSE IF ( IOSS > 0 ) THEN
  CALL ERROR_PROCESSING
END IF
```

13.4.2 Input/Output List

The input/output list of a file I/O statement specifies the entities whose values are to be read or written by the statement. An input list supplies one or more variables to receive values read. An output list supplies one or more expressions to be evaluated and sent to fields of an output record.

An *io-item* is one of: R13-9
- *input-item*
- *output-item*

An *input-item* is one of: R13-10
- *variable*
- *io-implied-do*

An *output-item* is one of: R13-11
- *expr*
- *io-implied-do*

An *io-implied-do* is: R13-12
(*io-item(s)*, *io-implied-do-control*)

An *io-implied-do-control* is: R13-13
index-variable = *scalar-num-expr*, *scalar-num-expr* [, *scalar-num-expr*]

- The *index-variable* of an *io-implied-do-control* must be a named scalar variable of type integer or real.
- In an *input-item*, an *io-implied-do* must contain only *input-item*s in its *io-item(s)* list. In an *output-item(s)* list, an *io-implied-do* must contain only output items in its *io-item(s)* list.

Copyright © 1989–1994 Thinking Machines Corporation

The following restrictions hold for input and output lists, as indicated:

- An *input-item* must not appear as, nor be associated with, the *index-variable* of any *io-implied-do* that contains the *input-item*.
- The *index-variable* of an *io-implied-do* that is contained within another *io-implied-do* must not be the same as, nor be associated with, the *index-variable* of the containing *io-implied-do*.
- Neither the whole name of an assumed-size array nor a section of an assumed-size array may appear as an I/O list item.
- An expression that forms an *output-item* must not reference a function that causes another I/O statement to be executed.
- Every entity of an output list whose value is to be transferred must be defined.

The loop initialization and execution of an *io-implied-do* is the same as for the *loop-construct* (Section 11.5).

If a whole array name appears as an item in an *input-item(s)* list or *output-item(s)* list, it is treated as if its elements, if any, were specified in array element order.

All values needed to determine which entities are specified by a particular I/O list item are evaluated at the beginning of the processing of that item. Also, all values are transmitted to or from the entities specified by a given list item prior to the processing of any succeeding list item. For example, in the `READ` statement

```
READ( N ) N, X(N)
```

the original value of `N` identifies the unit, but the newly read value of `N` is the subscript of `X`.

A `DO` index variable becomes defined and its iteration count established at the beginning of processing of the I/O list items that constitute the range of an *io-implied-do*.

Examples of output lists with an implied `DO`:

```
PRINT *, ( 'XYZZY ', I = 1, N ) ! writes N copies of 'XYZZY'

WRITE ( LP, FMT='(10F8.2)' ) ( LOG( A(I) ), I = 1, N+9, K ), G
! Writes the logarithm of selected elements of array A,
!   followed by G (which may be a scalar or an array).
```

For information about using `NAMELIST` to identify a set of variables to be read or written, see Sections 5.7 and 13.4.1.3.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

13.5 Sequential Data Transfer — External Files

The I/O statements described in the following sections transfer data to and from external files accessed under the *sequential* mode of access, processing records one after another from the beginning of a file to its end.

13.5.1 Formatted Sequential READ (External Files)

These `READ` statements read one or more records of an external ASCII character file, convert fields of those records to binary values, and store the converted values into variables designated in the input list. A format specification defines the conversion to be performed for each field.

A (formatted sequential) *read-stmt* is one of: R13-14

- `READ (` *io-control-spec(s)* `)` *[input-item(s)]*
- `READ` *format-identifier [* `,` *input-item(s)]*

A (formatted sequential) *io-control-spec* is one of: R13-15

- *[* `UNIT =` *] external-unit-identifier* ! required
- *[* `FMT =` *] format-identifier* ! required
- `END =` *label*
- `ERR =` *label*
- `IOSTAT =` scalar-integer-*variable*

An *external-unit-identifier* is one of: R13-16

- *unit-number*
- *

A (formatted sequential) *format-identifier* is one of: R13-17

- *label*
- *char-expr*
- scalar-integer-*variable*

An *input-item* is one of: R13-10

- *variable*
- *io-implied-do*

♦ A `UNIT=` specifier and a `FMT=` specifier are required in the first form of `READ` statement. Other syntactic restrictions on the form of an *io-control-spec(s)* list are given in Section 13.4.1.

The first form of `READ` statement reads from the specified unit number; the second form reads from the standard input device. The operation involves the following steps.

Copyright © 1989–1994 Thinking Machines Corporation

Identify Unit. A `UNIT=` specifier of `*` identifies the unit preconnected to the standard input file for formatted sequential access.

Establish Format. The `FMT=` specifier establishes formatted editing according to the given format specification.

Position File Prior to Transfer. The file is positioned at the beginning of the next record prior to data transfer, and this record becomes the current record. (When the file is first opened, it is positioned prior to the beginning of its first record, and there is no current record.)

If the file contains an endfile record, the file must not be positioned after the endfile record prior to data transfer.

Transfer Data. Data is transferred with editing from the current record to the entities specified by the input list (if any). The input items are processed in the order they appear in the input list. Format control is initiated and editing is performed as in Chapter 15.

> **Records Read.** The current record and possibly additional succeeding records are read. Additional records are read if the input list has not been satisfied and either (1) the format specification is exhausted, causing format control to revert within the specification, or (2) a slash edit descriptor is encountered in the format specification.
>
> **Record Underflow.** If the current record contains less data than is necessary to satisfy all the input items and the format specification, an error occurs.
>
> **Record Overflow.** If the current record contains more data than is necessary to satisfy all the input items and the associated format specification, the extra data is skipped over.
>
> **Null Input List.** If the input list is not specified and the format specification starts with a data edit descriptor, or is empty, a record is skipped over and no data is transferred.
>
> **Termination.** Data transfer stops when data has been transmitted to every item of the input list, when end of file is reached, or when an error occurs.

Position After Transfer. If no error occurs, the file is positioned after the last record read.

If an error occurs, the position of the file is indeterminate. No indication is given of which record could not be read, nor of how many input items were read, if any.

Update IOSTAT. If the statement contains an `IOSTAT=` specifier, the integer variable is set to zero if no error or end-of-file condition resulted from the operation, to –1 if end of

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

file was reached, or to a positive error code if an error occurred. See Section 13.8 for a list of the I/O status codes.

If the statement contains an ERR= specifier, control is transferred to the specified label if an error occurs.

If the statement contains an END= specifier, control is transferred to the specified label if end of file is reached.

Examples of formatted sequential READ statements:

```
      READ( 10, 100 )   INC, EPSILON
100   FORMAT( I3, F9.3 )

      READ 100, INC, EPSILON

      READ "(I3, F9.3)", INC, EPSILON
```

13.5.2 Formatted Sequential WRITE and PRINT (External Files)

These WRITE and PRINT statements evaluate a list of expressions, convert the resulting values to character form, and transfer the converted results to one or more records of an external file. A format specification defines the conversion to be performed for each expression.

A (formatted sequential) *write-stmt* is: R13-18

`WRITE (` *io-control-spec(s)* `)` *[output-item(s)]*

A (formatted sequential) *print-stmt* is: R13-19

`PRINT` *format-identifier [, output-item(s)]*

A (formatted sequential) *io-control-spec* is one of: R13-20

- *[* `UNIT =` *] external-unit-identifier* ! required
- *[* `FMT =` *] format-identifier* ! required
- `ERR =` *label*
- `IOSTAT =` scalar-integer-*variable*

An *external-unit-identifier* is one of: R13-16

- *unit-number*
- *

A (formatted sequential) *format-identifier* is one of: R13-21

- *label*
- *char-expr*
- scalar-integer-*variable*

Copyright © 1989–1994 Thinking Machines Corporation

An *output-item* is one of: R13-11

- *expr*
- *io-implied-do*

♦ A `UNIT=` specifier and a `FMT=` specifier are required in this form of `WRITE` statement. Other syntactic restrictions on the form of an *io-control-spec(s)* list are given in Section 13.4.1.

These `WRITE` statements write to the specified unit number; the `PRINT` statements write to the standard output device. The operation involves the following steps.

Identify Unit. A `UNIT=` specifier of `*` identifies the unit preconnected to the standard output file for formatted sequential access. The `PRINT` statement always specifies the standard output file.

Establish Format. The `FMT=` specifier establishes formatted editing according to the given format specification.

Position File Prior to Transfer. A new record is created and becomes the current and last record of the file.

If the file contains an endfile record, the file must not be positioned after the endfile record prior to data transfer. This statement may be used to write over an endfile record, provided that the file is positioned just before the endfile record when the statement is executed.

Transfer Data. Data is transferred with editing from the entities specified by the output list (if any) to the current record. The output items are processed in the order they appear in the output list. Format control is initiated and editing is performed as described in Chapter 15.

Records Written. The current record and possibly additional succeeding records are written. Additional records are written if (1) the number of output items exceeds the number of data edit descriptors in the format specification, causing format control to revert within the specification, or (2) a slash edit descriptor is encountered in the format specification.

The length of records written by this statement depends upon the number and type of output items written. Records may differ in length depending on the number and kind of edit descriptors used when they are written.

Record Underflow. If the number of output items is fewer than the number of data edit descriptors in the format specification, the extra edit descriptors are not used, and a truncated record is written.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Record Overflow. If the number of output items exceeds the number of data edit descriptors in the format specification, the current record and additional records are written as described above.

Null Output List. If the output list is not specified and the format specification starts with a data edit descriptor, or is empty, a blank record is written.

Termination. Data transfer stops when data has been transmitted from every item of the output list, or when an error occurs.

Position After Transfer. If no error occurs, the file is positioned after the last record written.

If an error occurs, the position of the file is indeterminate. No indication is given of which record could not be written, nor of how many output items were not written.

Update IOSTAT. If the statement contains an `IOSTAT=` specifier, the integer variable is set to zero if no error condition resulted from the operation, or to a positive error code if an error occurred.

If the statement contains an `ERR=` specifier, control is transferred to the specified label if an error occurs.

13.5.2.1 Printing of Formatted Records

Information in formatted records is handled specially by some printing devices. If a formatted record is printed, the first character of the record is not printed. The remaining characters of the record, if any, are printed in one line beginning at the left margin.

The first character of such a record determines vertical spacing as follows:

Character	Vertical Spacing Before Printing
blank	one line
`0`	two lines
`1`	to first line of next page
`+`	no advance (overwrite)

If there are no characters in the record, the vertical spacing is one line and no characters other than blank are printed in that line.

In CM Fortran there are no such printing devices, so all characters are transferred to the file. The UNIX command `lpr` with the `-f` option allows such files to be printed.

Copyright © 1989–1994 Thinking Machines Corporation

13.5.3 Unformatted Sequential READ (External Files)

These READ statements read a single record of an external file and transfer fields of binary data from the record without conversion into variables designated in the input list.

An (unformatted sequential) *read-stmt* is: R13-22

READ (*io-control-spec(s)*) *[input-item(s)]*

An (unformatted sequential) *io-control-spec* is one of: R13-23

- *[* UNIT = *]* *unit-number* ! required
- END = *label*
- ERR = *label*
- IOSTAT = scalar-integer-*variable*

An *input-item* is one of: R13-10

- *variable*
- *io-implied-do*

♦ A UNIT= specifier is required in this form of READ statement. Other syntactic restrictions on the form of an *io-control-spec(s)* list are given in Section 13.4.1.

These READ statements read from the specified unit number. The operation involves the following steps.

Identify Unit. The unit specified must be connected (explicitly) for unformatted sequential access to an existing external file when execution of the statement begins.

Establish Format. The lack of a FMT= specifier establishes unformatted data transfer.

Position File Prior to Transfer. The file is positioned at the beginning of the next record prior to data transfer, and this record becomes the current record. (When the file is first opened, it is positioned prior to the beginning of its first record.)

If the file contains an endfile record, the file must not be positioned after the endfile record prior to the read.

Transfer Data. Data is transferred without editing from the current record to the entities specified by the input list (if any). The input items are processed in the order they appear in the input list.

Records Read. A single record (the current record) is read.

The type of each field in the record must agree with the type of the corresponding item in the input list, except that one complex field may correspond to two real input items, or two real fields may correspond to one complex input item (and the precision of the input

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

items must match the precision of the fields). If an item in the input list is of type character, the length of the character item must agree with the length of the character field read.

Record Underflow. If the current record contains less data than is necessary to satisfy all the input items, an error occurs.

Record Overflow. If the current record contains more data than is necessary to satisfy all the input items, the extra data is skipped over.

Null Input List. If the input list is not specified, no data is transferred.

Termination. Data transfer stops when data has been transmitted to every item of the input list, when end of file is reached, or when an error occurs.

Position After Transfer. If no error occurs, the file is positioned after the last record read.

If an error occurs, the position of the file is indeterminate. No indication is given of how many input items were read, if any.

Update IOSTAT. If the statement contains an `IOSTAT=` specifier, the integer variable is set to zero if no error or end-of-file condition resulted from the operation, to `-1` if end of file was reached, or to a positive error code if an error occurred.

If the statement contains an `ERR=` specifier, control is transferred to the specified label if an error occurs.

If the statement contains an `END=` specifier, control is transferred to the specified label if end of file is reached.

13.5.4 Unformatted Sequential WRITE (External Files)

These `WRITE` statements evaluate a list of expressions and transfer the resulting binary values without conversion to a single record of an external file.

An (unformatted sequential) *write-stmt* is: R13-24

`WRITE (` *io-control-spec(s)* `)` *[output-item(s)]*

An (unformatted sequential) *io-control-spec* is one of: R13-25

- *[* `UNIT =` *] unit-number* ! required
- `ERR =` *label*
- `IOSTAT =` scalar-integer-*variable*

Copyright © 1989–1994 Thinking Machines Corporation

An *output-item* is one of: R13-11

- *expr*
- *io-implied-do*

♦ A `UNIT=` specifier is required in this form of `WRITE` statement. Other syntactic restrictions on the form of an *io-control-spec(s)* list are given in Section 13.4.1.

The `WRITE` statement writes to the specified unit number. The operation involves the following steps.

Identify Unit. The unit specified must be connected (explicitly) for unformatted sequential access to an external file when execution of the statement begins.

Establish Format. The lack of a `FMT=` specifier establishes unformatted data transfer.

Position File Prior to Transfer. A new record is created and becomes the current and last record of the file.

If the file contains an endfile record, the file must not be positioned after the endfile record prior to data transfer. This statement may be used to write over an endfile record, provided that the file is positioned just before the endfile record when the statement is executed.

Transfer Data. Data is transferred without editing from the entities specified by the output list (if any) to the current record. The output items are processed in the order they appear in the output list.

Records Written. A single record (the current record) is written.

The length of a record is defined by the number of bytes required to represent the output items, and records in a file may differ in length.

Record Underflow. N/A

Record Overflow. N/A

Null Output List. No data is transferred.

Termination. Data transfer stops when data has been transmitted from every item of the output list, or when an error occurs.

Position After Transfer. If no error occurs, the file is positioned after the last record written, which becomes the last record of the file.

If an error occurs, the position of the file is indeterminate. No indication is given of which record could not be written, nor of how many output items were not written.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Update IOSTAT. If the statement contains an IOSTAT= specifier, the integer variable is set to zero if no error condition resulted from the operation, or to a positive error code if an error occurred.

If the statement contains an ERR= specifier, control is transferred to the specified label if an error occurs.

13.5.5 List-Directed READ (External Files)

These READ statements read one or more records of an external ASCII character file with a standard format, convert characters of the record to binary values, and store the converted values into variables designated in the input list. The type of each variable in the input list of the statement determines the conversion to be performed for that variable.

A (list-directed external) *read-stmt* is one of: R13-26

- READ (*io-control-spec(s)*) *[input-item(s)]*
- READ * *[, input-item(s)]*

A (list-directed external) *io-control-spec* is one of: R13-27

- *[* UNIT = *]* *external-unit-identifier* ! required
- *[* FMT = *]* * ! required
- END = *label*
- ERR = *label*
- IOSTAT = scalar-integer-*variable*

An *external-unit-identifier* is one of: R13-16

- *unit-number*
- *

An *input-item* is one of: R13-10

- *variable*
- *io-implied-do*

♦ A UNIT= specifier and a FMT= specifier are required in this form of READ statement. Other syntactic restrictions on the form of an *io-control-spec(s)* list are given in Section 13.4.1.

The first form of READ statement reads from the specified unit number; the second form reads from the standard input device. The operation involves the following steps.

Identify Unit. A UNIT= specifier of * identifies the unit preconnected to the standard input file for formatted sequential access.

Copyright © 1989–1994 Thinking Machines Corporation

Establish Format. The FMT= specifier of * establishes list-directed editing.

Position File Prior to Transfer. The file is positioned at the beginning of the next record prior to data transfer, and this record becomes the current record. (When the file is first opened, it is positioned prior to the beginning of its first record, and there is no current record.)

If the file contains an endfile record, the file must not be positioned after the endfile record prior to data transfer.

Transfer Data. Data is transferred with list-directed editing from the current record to the entities specified by the input list (if any). The input items are processed in the order they appear in the input list. List-directed format editing proceeds as described in Chapter 15.

Records Read. The current record and possibly additional succeeding records are read. Additional records are read as necessary to satisfy all items of the input list. (The end of a record is treated as a blank character except when it appears in a character constant.)

Record Underflow. If the current record contains less data than is necessary to satisfy all the input items, additional records are read as necessary to satisfy all items of the input list.

If the input list has not been satisfied when a slash separator is encountered in a record or when an end-of-file record is encountered, execution of the READ statement is terminated, and the remaining items of the input list are unaltered.

Record Overflow. If the current record contains more data than is necessary to satisfy all the items of the input list, the extra data is skipped over.

Null Input List. If the input list is not specified, no data is transferred.

Termination. Data transfer stops when data has been transmitted to every item of the input list, when a slash separator is encountered in the file, when end of file is reached, or when an error occurs.

Position After Transfer. If no error occurs, the file is positioned after the last record read.

If an error occurs, the position of the file is indeterminate. No indication is given of which record could not be read, nor of how many input items were read, if any.

Update IOSTAT. If the statement contains an IOSTAT= specifier, the integer variable becomes defined with a zero if no error condition or end-of-file condition exists, a –1 if end of file is reached, and a positive error code if an error occurs.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

If the statement contains an ERR= specifier, control is transferred to the specified label if an error occurs.

If the statement contains an END= specifier, control is transferred to specified label if end of file is reached.

13.5.6 List-Directed WRITE and PRINT (External Files)

These WRITE and PRINT statements evaluate a list of expressions, convert the resulting values to a standard character form, and transfer the converted results to a single record of an external character file. The type of each expression in the output list of the statement determines the conversion to be performed for that expression value.

A (list-directed external) *write-stmt* is: R13-28

WRITE (*io-control-spec(s)*) *[output-item(s)]*

A (list-directed external) *print-stmt* is: R13-29

PRINT * *[, output-item(s)]*

A (list-directed external) *io-control-spec* is one of: R13-30

- *[* UNIT = *]* *external-unit-identifier* ! required
- *[* FMT = *]* * ! required
- ERR = *label*
- IOSTAT = scalar-integer-*variable*

An *external-unit-identifier* is one of: R13-16

- *unit-number*
- *

An *output-item* is one of: R13-11

- *expr*
- *io-implied-do*

♦ A UNIT= specifier and a FMT= specifier are required in this form of WRITE statement. Other syntactic restrictions on the form of an *io-control-spec(s)* list are given in Section 13.4.1.

This WRITE statement writes to the specified unit number; the PRINT statement writes to the standard output device. The operations involve the following steps.

Identify Unit. A UNIT= specifier of * identifies the unit preconnected to the standard output file for formatted sequential access. The PRINT statement always specifies the standard output file.

Copyright © 1989–1994 Thinking Machines Corporation

Establish Format. The FMT= specifier of * establishes list-directed editing.

Position File Prior to Transfer. A new record is created and becomes the current and last record of the file.

If the file contains an endfile record, the file must not be positioned after the endfile record prior to data transfer. This statement may be used to write over an endfile record, provided that the file is positioned just before the endfile record when the statement is executed.

Transfer Data. Data is transferred with list-directed editing from the entities specified by the output list (if any) to the current record. The output items are processed in the order they appear in the output list. List-directed format editing proceeds as described in Chapter 15.

Records Written. A single record (the current record) is written.

The length of a record is defined by the number of characters required to represent all the output items. The records written may differ in length depending on the type and number of output items in the output list and the data separators between them.

Record Underflow. N/A

Record Overflow. N/A

Null Output List. No data is transferred.

Termination. Data transfer stops when data has been transmitted from every item of the output list, or when an error occurs.

Position After Transfer. If no error occurs, the file is positioned after the last record written, which becomes the last record of the file.

If an error occurs, the position of the file is indeterminate. No indication is given of how many output items were not written.

Update IOSTAT. If the statement contains an IOSTAT= specifier, the integer variable is set to zero if no error condition resulted from the operation, or to a positive error code if an error occurred.

If the statement contains an ERR= specifier, control is transferred to the specified label if an error occurs.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

13.6 Sequential Data Transfer — Internal Files

The I/O statements described in the following sections transfer data to and from internal files.

13.6.1 Formatted Sequential READ (Internal Files)

These READ statements read one or more records of a scalar or array character variable (each array element is considered a record), convert fields in each record to binary values, and store the converted values into variables designated in the input list. A format specification specifies the conversion to be performed for each field.

A (formatted sequential internal) *read-stmt* is: R13-31

READ (*io-control-spec(s)*) *[input-item(s)]*

A (formatted sequential internal) *io-control-spec* is one of: R13-32

- *[* UNIT = *] scalar-char-variable* ! required
- *[* FMT = *] format-identifier* ! required
- END = *label*
- ERR = *label*
- IOSTAT = scalar-integer-*variable*

A (formatted sequential internal) *format-identifier* is one of: R13-33

- *label*
- *char-expr*
- scalar-integer-*variable*

An *input-item* is one of: R13-10

- *variable*
- *io-implied-do*

♦ A UNIT= specifier and a FMT= specifier are required in this form of READ statement. Other syntactic restrictions on the form of an *io-control-spec(s)* list are given in Section 13.4.1.

These READ statements read from the specified unit number. The operation involves the following steps.

Identify Unit. The UNIT= specifier identifies a scalar or array character variable as the internal file to be read.

Establish Format. The FMT= specifier establishes formatted editing according to the given format specification.

Copyright © 1989–1994 Thinking Machines Corporation

Position File Prior to Transfer. An internal file is always positioned at the beginning of its first record prior to data transfer, and this record becomes the current record.

Transfer Data. Data is transferred with editing from records of the file to entities specified by the input list (if any), starting with the first record of the file. The input items are processed in the order they appear in the input list. Format control is initiated and editing is performed as described in Chapter 15.

Records Read. The current record and possibly additional succeeding records are read. Additional records are read if the input list has not been satisfied and either (1) the format specification is exhausted, causing format control to revert within the specification, or (2) a slash edit descriptor is encountered in the format specification.

Record Underflow. If the current record contains less data than is necessary to satisfy all the input items and the format specification, an error occurs.

Record Overflow. If the current record contains more data than is necessary to satisfy all the input items and the associated format specification, the extra data is ignored.

Null Input List. If the input list is not specified and the format specification starts with a data edit descriptor, or is empty, a record is skipped over, and no data is transferred.

Termination. Data transfer stops when data has been transmitted to every item of the input list, when end of file is reached, or when an error occurs.

Position After Transfer. After data transfer, the file is positioned at the beginning of its first record.

If an error occurs, no indication is given of how many input items were read, if any.

Update IOSTAT. If the statement contains an `IOSTAT=` specifier, the integer variable is set to zero if no error or end-of-file condition resulted from the operation, to –1 if end of file was reached, or to a positive error code if an error occurred.

If the statement contains an `ERR=` specifier, control is transferred to the specified label if an error occurs.

If the statement contains an `END=` specifier, control is transferred to the specified label if end of file is reached.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

13.6.2 Formatted Sequential WRITE (Internal Files)

These WRITE statements evaluate a list of expressions, convert the resulting values to character form, and transfer the converted results to one or more records of an internal file variable. A format specification specifies the conversion to be performed for each expression.

A (formatted sequential internal) *write-stmt* is: R13-34
WRITE (*io-control-spec(s)*) *[output-item(s)]*

A (formatted sequential internal) *io-control-spec* is one of: R13-35
- *[* UNIT = *] scalar-char-variable* ! required
- *[* FMT = *] format-identifier* ! required
- ERR = *label*
- IOSTAT = scalar-integer-*variable*

A (formatted sequential internal) *format-identifier* is one of: R13-36
- *label*
- *char-expr*
- scalar-integer-*variable*

An *output-item* is one of: R13-11
- *expr*
- *io-implied-do*

♦ A UNIT= specifier and a FMT= specifier are required in this form of WRITE statement. Other syntactic restrictions on the form of an *io-control-spec(s)* list are given in Section 13.4.1.

These WRITE statements write to the specified unit number. The operation involves the following steps.

Identify Unit. The UNIT= specifier identifies a scalar or array character variable as the internal file to be written.

Establish Format. The FMT= specifier establishes formatted editing according to the given format specification.

A format specification that is a substring of the internal file variable must not be specified. (More generally, the format specification must not be storage-associated in any way with the specified internal file variable.)

Position File Prior to Transfer. An internal file is always positioned at the beginning of its first record prior to the WRITE, and this record becomes the current record.

Copyright © 1989–1994 Thinking Machines Corporation

Transfer Data. Data is transferred with editing from the entities specified by the output list (if any) to the current record. The output items are processed in the order they appear in the output list. Format control is initiated and editing is performed as described in Chapter 15.

Records Written. The current record and possibly additional succeeding records are written. Additional records are written if (1) the number of output items exceeds the number of data edit descriptors in the format specification, causing format control to revert within the specification, or (2) a slash edit descriptor is encountered in the format specification.

The length of records written is the length of the scalar character variable or the length of (an element of) the character array forming the internal file variable.

After execution of this `WRITE` statement, records following the last record to which data was written are unchanged.

Record Underflow. If the number of output items is fewer than the number of data edit descriptors in the format specification, the extra edit descriptors are not used, and trailing blank characters are added to fill the record.

Record Overflow. If a record is not large enough to hold all the converted items, an error occurs.

Null Output List. If the output list is not specified, a blank record is written.

Termination. Data transfer stops when data has been transmitted from every item of the output list, when every record has been written, or when an error occurs.

Position After Transfer. After data transfer, the file remains positioned at the beginning of its first record.

If an error occurs, no indication is given of which record could not be written, nor of how many output items were not written.

Update IOSTAT. If the statement contains an `IOSTAT=` specifier, the integer variable is set to zero if no error condition resulted from the operation, or to a positive error code if an error occurred.

If the statement contains an `ERR=` specifier, control is transferred to the specified label if an error occurs.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

13.6.3 List-Directed READ (Internal Files)

These READ statements read character sequences from a scalar or array character variable with a standard format, convert substrings of the variable to binary values, and store the converted values into program variables designated in the input list. The type of each input variable determines the conversion to be performed for that variable.

A (list-directed internal) *read-stmt* is: R13-37
READ (*io-control-spec(s)*) *[input-item(s)]*

A (list-directed internal) *io-control-spec* is one of: R13-38
- *[* UNIT = *] scalar-char-variable* ! required
- *[* FMT = *]* * ! required
- END = *label*
- ERR = *label*
- IOSTAT = scalar-integer-*variable*

An *input-item* is one of: R13-10
- *variable*
- *io-implied-do*

♦ A UNIT= specifier and a FMT= specifier are required in this form of READ statement. Other syntactic restrictions on the form of an *io-control-spec(s)* list are given in Section 13.4.1.

These READ statements read from the specified unit number. The operation involves the following steps.

Identify Unit. The UNIT= specifier identifies a scalar or array character variable as the internal file to be read.

Establish Format. The FMT= specifier of * establishes list-directed editing.

Position File Prior to Transfer. An internal file is always positioned at the beginning of its first record prior to the READ, and this record becomes the current record.

Transfer Data. Data is transferred with list-directed editing from the current record to the entities specified by the input list (if any). The input items are processed in the order they appear in the input list. List-directed format editing proceeds as described in Chapter 15.

Records Read. The current record and possibly additional succeeding records are read. If the internal file is an array, additional records are read as necessary to satisfy all items of the input list. (The end of a record is treated as a blank character except when it appears in a character constant.)

Copyright © 1989–1994 Thinking Machines Corporation

If the input list has not been satisfied when a slash separator is encountered in a record or when an end-of-file record is encountered, execution of the READ statement is terminated, and the remaining items of the input list are unaltered.

Record Underflow. If the current record contains less data than is necessary to satisfy all the input items, additional records are read as described above.

If the input list has not been satisfied when a slash separator is encountered in a record or the end of file is reached, execution of the READ statement is terminated, and the remaining items of the input list are unaltered.

Record Overflow. If the current record contains more data than is necessary to satisfy all the input items, the extra data is ignored.

Null Input List. If the input list is not specified, no data is transferred.

Termination. Data transfer stops when data has been transmitted to every item of the input list, when a slash separator is encountered in the file, when end of file is reached, or when an error occurs.

Position after Transfer. After data transfer, the file is positioned at the beginning of its first record.

If an error occurs, no indication is given of which record could not be read, nor of how many input items were read, if any.

Update IOSTAT. If the statement contains an IOSTAT= specifier, the integer variable is set to zero if no error or end-of-file condition resulted from the operation, to –1 if end of file was reached, or to a positive error code if an error occurred.

If the statement contains an ERR= specifier, control is transferred to the specified label if an error occurs.

If the statement contains an END= specifier, control is transferred to the specified label if end of file is reached.

13.6.4 List-Directed WRITE (Internal Files)

These WRITE statements evaluate a list of expressions, convert the resulting values to a standard character form, and transfer the converted results to one or more records of an internal file variable. The type of each expression in the output list of the statement determines the conversion to be performed for that expression value.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

A (list-directed internal) *write-stmt* is: R13-39

`WRITE (` *io-control-spec(s)* `)` *[output-item(s)]*

A (list-directed internal) *io-control-spec* is one of: R13-40

- *[* `UNIT =` *]* *scalar-char-variable* ! required
- *[* `FMT =` *]* `*` ! required
- `ERR =` *label*
- `IOSTAT =` scalar-integer-*variable*

An *output-item* is one of: R13-11

- *expr*
- *io-implied-do*

♦ A `UNIT=` specifier and a `FMT=` specifier are required in this form of `WRITE` statement. Other syntactic restrictions on the form of an *io-control-spec(s)* list are given in Section 13.4.1.

These `WRITE` statements write to the specified unit number. The operation involves the following steps.

Identify Unit. The `UNIT=` specifier identifies a scalar or array character variable as the internal file to be written.

Establish Format. The `FMT=` specifier of `*` establishes list-directed editing.

Position File Prior to Transfer. An internal file is always positioned at the beginning of its first record prior to the `WRITE`, and this record becomes the current record.

Transfer Data. Data is transferred with list-directed editing from the entities specified by the output list (if any) to the current record. The output items are processed in the order they appear in the output list. List-directed format editing proceeds as described in Chapter 15.

Records Written. The current record and possibly additional succeeding records are written.

If the file variable is an array, this statement writes as many records as are required to hold the converted output list items, up to the total number of records in the file (elements in the array).

The characters written to a record by this statement consist of a sequence of values and value separators of commas, slashes, blanks, and parentheses, as described in Chapter 15.

Copyright © 1989–1994 Thinking Machines Corporation

The length of a record is the length of the scalar character variable or the length of (an element of) the character array forming the internal file variable.

After execution of this `WRITE` statement, records following the last record to which data was written are unchanged.

Record Underflow. If the number of characters required to represent the converted output items does not fill a record, trailing blank characters are added to fill the record.

Record Overflow. If a record is not large enough to hold all the converted items, a new record is started for any noncharacter item whose converted result would exceed the record capacity. For character items, as much as can be put in a record is written there, and the remainder is written at the beginning of the next record.

If the record length is insufficient to hold any noncharacter item, an error occurs.

Null Output List. If the output list is not specified, a blank record is written.

Termination. Data transfer stops when data has been transmitted from every item of the output list, when every record has been written, or when an error occurs.

Position after Transfer. After data transfer, the file remains positioned at the beginning of its first record.

Update IOSTAT. If the statement contains an `IOSTAT=` specifier, the integer variable is set to zero if no error condition resulted from the operation, or to a positive error code if an error occurred.

If the statement contains an `ERR=` specifier, control is transferred to the specified label if an error occurs.

13.7 Direct Access Data Transfer — External Files

The I/O statements described in the following sections transfer data to and from external files accessed under the *direct* mode of access, processing selected records of a file in arbitrary order.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

13.7.1 Formatted Direct Access READ (External Files)

These READ statements read one or more specified records of an external ASCII character file, convert fields of the record to binary values, and store the converted values into variables designated in the input list. The first record to be read is identified by its record number. A format specification specifies the conversion to be performed for each field.

A (formatted direct access) *read-stmt* is: R13-41
READ (*io-control-spec(s)*) *[input-item(s)]*

A (formatted direct access) *io-control-spec* is one of: R13-42
- *[* UNIT = *] unit-number* ! required
- *[* FMT = *] format-identifier* ! required
- REC = *record-number* ! required
- ERR = *label*
- IOSTAT = scalar-integer-*variable*

A (formatted direct access) *format-identifier* is one of: R13-43
- *label*
- *char-expr*
- scalar-integer-*variable*

An *input-item* is one of: R13-10
- *variable*
- *io-implied-do*

♦ A UNIT= specifier, a FMT= specifier, and a REC= specifier are required in this form of READ statement. Other syntactic restrictions on the form of an *io-control-spec(s)* list are given in Section 13.4.1.

These READ statements read from the specified unit number. The operation involves the following steps.

Identify Unit. The unit specified must be connected for formatted direct access to an external file when execution of the statement begins.

Establish Format. The FMT= specifier establishes formatted editing according to the given format specification.

Position File Prior to Transfer. The file is positioned at the beginning of the record specified by the REC= specifier, and this record becomes the current record.

Copyright © 1989–1994 Thinking Machines Corporation

Transfer Data. Data is transferred with editing from the current record to the entities specified by the input list (if any). The input items are processed in the order they appear in the input list. Format control is initiated and editing is performed as described in Chapter 15.

Records Read. The current record and possibly additional succeeding records are read. Additional records are read if the input list has not been satisfied and either (1) the format specification is exhausted, causing format control to revert within the specification, or (2) a slash edit descriptor is encountered in the format specification.

An attempt to read a record that has not previously been written causes all input list items to become undefined.

Record Underflow. If the current record contains less data than is necessary to satisfy all the input items and the format specification, an error occurs.

Record Overflow. If the current record contains more data than is necessary to satisfy all the input items and the associated format specification, the extra data is skipped over.

Null Input List. If the input list is not specified and the format specification starts with a data edit descriptor, or is empty, the record number is increased by one and no data transferred.

Termination. Data transfer stops when data has been transmitted to every item of the input list, or when an error occurs.

Position After Transfer. If no error occurs, the file is positioned after the last record read. An `INQUIRE` statement (with a `NEXTREC=` specifier) can be used to determine the next record number.

If an error occurs, the position of the file is indeterminate. No indication is given of which record could not be read, nor of how many input items were read, if any.

Update IOSTAT. If the statement contains an `IOSTAT=` specifier, the integer variable is set to zero if no error condition resulted from the operation, or to a positive error code if an error occurred.

If the statement contains an `ERR=` specifier, control is transferred to the specified label if an error occurs.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

13.7.2 Formatted Direct Access WRITE (External Files)

These WRITE statements evaluate a list of expressions, convert the resulting values to character form, and transfer the converted results to one or more records of an external character file. The first record to be written is identified by its record number. A format specification specifies the conversion to be performed for each expression.

A (formatted direct access) *write-stmt* is: R13-44

WRITE (*io-control-spec(s)*) *[output-item(s)]*

A (formatted direct access) *io-control-spec* is one of: R13-45

- *[* UNIT = *] unit-number* ! required
- *[* FMT = *] format-identifier* ! required
- REC = *record-number* ! required
- ERR = *label*
- IOSTAT = scalar-integer-*variable*

A (formatted direct access) *format-identifier* is one of: R13-46

- *label*
- *char-expr*
- scalar-integer-*variable*

An *output-item* is one of: R13-11

- *expr*
- *io-implied-do*

♦ A UNIT= specifier, a FMT= specifier, and a REC= specifier are required in this form of WRITE statement. Other syntactic restrictions on the form of an *io-control-spec(s)* list are given in Section 13.4.1.

These WRITE statements write to the specified unit number. The operation involves the following steps.

Identify Unit. The unit specified must be connected (explicitly) for formatted direct access to an external file when execution of the statement begins.

Establish Format. The FMT= specifier establishes formatted editing according to the given format specification.

Position File Prior to Transfer. The file is positioned at the beginning of the record specified by the REC= specifier, and this record becomes the current record.

Transfer Data. Data is transferred with editing from the entities specified by the output list (if any) to the current record. Format control is initiated and editing is performed as

Copyright © 1989–1994 Thinking Machines Corporation

described in Chapter 15. The output items are processed in the order they appear in the output list.

Records Written. The current record and possibly additional succeeding records are written. Additional records are written if (1) the number of output items exceeds the number of data edit descriptors in the format specification, causing format control to revert within the specification, or (2) a slash edit descriptor is encountered in the format specification.

The length of every record in the file is designated by the `RECL=` specifier of the `OPEN` statement.

Record Underflow. If the number of output items is fewer than the number of data edit descriptors in the format specification, the extra edit descriptors are not used, and trailing blank characters are added to fill the record.

Record Overflow. If the total number of characters specified by the output list and format specification are greater than the record length for the file, an error occurs.

Null Output List. If the output list is not specified and the format specification starts with a data edit descriptor, or is empty, a blank record is written and the record number is increased by one.

Termination. Data transfer stops when data has been transmitted from every item of the output list, or when an error occurs.

Position After Transfer. If no error occurs, the file is positioned after the last record written. An `INQUIRE` statement (with a `NEXTREC=` specifier) can be used to determine the next record number.

If an error occurs, the position of the file is indeterminate. No indication is given of which record could not be written, nor of how many output items were written, if any.

Update IOSTAT. If the statement contains an `IOSTAT=` specifier, the integer variable is set to zero if no error condition resulted from the operation, or to a positive error code if an error occurred.

If the statement contains an `ERR=` specifier, control is transferred to the specified label if an error occurs.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

13.7.3 Unformatted Direct Access READ (External Files)

These READ statements read a specified record of an external file and transfer fields of binary data from the record without conversion into variables designated in the input list. The record to be read is identified by its record number.

An (unformatted direct access) *read-stmt* is: R13-47
READ (*io-control-spec(s)*) *[input-item(s)]*

An (unformatted direct access) *io-control-spec* is one of: R13-48
- *[* UNIT = *]* *unit-number* ! required
- REC = *record-number* ! required
- ERR = *label*
- IOSTAT = scalar-integer-*variable*

An *input-item* is one of: R13-10
- *variable*
- *io-implied-do*

♦ A UNIT= specifier and a REC= specifier are required in this form of READ statement. Other syntactic restrictions on the form of an *io-control-spec(s)* list are given in Section 13.4.1.

These READ statements read from the specified unit number. The operation involves the following steps.

Identify Unit. The unit specified must be connected (explicitly) for unformatted direct access to an existing external file when execution of the statement begins.

Establish Format. The lack of a FMT= specifier establishes unformatted data transfer.

Position File Prior to Transfer. The file is positioned at the beginning of the record specified by the REC= specifier, and this record becomes the current record.

Transfer Data. Data is transferred without editing from the current record to the entities specified by the input list (if any). The input items are processed in the order they appear in the input list.

Records Read. A single record (the current) is read.

The type of each field in the record must agree with the type of the corresponding item in the input list, except that one complex field may correspond to two real input items, or two real fields may correspond to one complex input item (and the precision of the input

Copyright © 1989–1994 Thinking Machines Corporation

items must match the precision of the fields). If an item in the input list is of type character, the length of the character item must agree with the length of the character field read.

An attempt to read a record that has not previously been written causes all input list items to become undefined.

Record Underflow. If the current record contains less data than is necessary to satisfy the items of the list, an error occurs.

Record Overflow. If the current record contains more data than is necessary to satisfy all the input items, the extra data is skipped over.

Null Input List. If the input list is not specified, the record number is increased by one and no data is transferred.

Termination. Data transfer stops when data has been transmitted to every item of the input list, or when an error occurs.

Position After Transfer. If no error occurs, the file is positioned after the record read. An `INQUIRE` statement (with a `NEXTREC=` specifier) can be used to determine the next record number.

If an error occurs, the position of the file is indeterminate. No indication is given of how many input items were read, if any.

Update IOSTAT. If the statement contains an `IOSTAT=` specifier, the integer variable is set to zero if no error condition resulted from the operation, or to a positive error code if an error occurred.

If the statement contains an `ERR=` specifier, control is transferred to the specified label if an error occurs.

13.7.4 Unformatted Direct Access WRITE (External Files)

These `WRITE` statements evaluate a list of expressions and transfer the resulting binary values without conversion to a single specified record of an external file. The record to be written is identified by its record number.

An (unformatted direct access) *write-stmt* is: R13-49

`WRITE (` *io-control-spec(s)* `)` *[output-item(s)]*

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

An (unformatted direct access) *io-control-spec* is one of: R13-50

- *[* `UNIT` = *]* *unit-number* ! required
- `REC` = *record-number* ! required
- `ERR` = *label*
- `IOSTAT` = scalar-integer-*variable*

An *output-item* is one of: R13-11

- *expr*
- *io-implied-do*

♦ A `UNIT`= specifier and a `REC`= specifier are required in this form of `WRITE` statement. Other syntactic restrictions on the form of an *io-control-spec(s)* list are given in Section 13.4.1.

These `WRITE` statements write to the specified unit number. The operation involves the following steps.

Identify Unit. The unit specified must be connected (explicitly) for unformatted direct access to an external file when execution of the statement begins.

Establish Format. The lack of a `FMT`= specifier establishes unformatted data transfer.

Position File Prior to Transfer. The file is positioned at the beginning of the record specified by the `REC`= specifier, and this record becomes the current record.

Transfer Data. Data is transferred without editing from the entities specified by the output list (if any) to the current record. The output items are processed in the order they appear in the output list.

Records Written. A single record (the current record) is written.

The length of every record in the file is designated by the `RECL`= specifier of the `OPEN` statement.

Record Underflow. If the values specified by the output list do not fill the record, the remainder of the record is undefined.

Record Overflow. If the total amount of data specified by the output list is greater than can be contained in a record, an error occurs.

Null Output List. If the output list is not specified, the internal record number is increased by one and no data is transferred.

Copyright © 1989–1994 Thinking Machines Corporation

Termination. Data transfer stops when data has been transmitted from every item of the output list, or when an error occurs.

Position After Transfer. If no error occurs, the file is positioned after the last record written. An `INQUIRE` statement (with a `NEXTREC=` specifier) can be used to determine the next record number.

If an error occurs, the position of the file is indeterminate. No indication is given of which record could not be written, nor of how many output items were written, if any.

Update IOSTAT. If the statement contains an `IOSTAT=` specifier, the integer variable is set to zero if no error condition resulted from the operation, or to a positive error code if an error occurred.

If the statement contains an `ERR=` specifier, control is transferred to the specified label if an error occurs.

13.8 Error and End-of-File Conditions

If an error condition occurs during execution of an I/O statement that contains neither an `IOSTAT` nor an `ERR=` specifier, or if an end-of-file condition occurs during execution of a `READ` statement that contains neither an `IOSTAT=` specifier nor an `END=` specifier, execution of the program is terminated. Table 10 lists the values that may be returned as status codes using the `ERR=` specifier in an I/O statement, along with a short description of the condition associated with each status code.

An I/O data transfer statement terminates when any of the following conditions are met:

- All elements of the *input-item(s)* list or *output-item(s)* list have been read or written, with or without editing, from or to the specified file.
- An error condition is detected.
- An end-of-file condition is detected.
- A slash is detected as a value separator in the record being read during list-directed input.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Table 10. Summary of run-time I/O status codes.

Error #	Message Text
–1	ENDFIL: end of file
00	FORTRAN STOP
01	NOTFORSPE: not FORTRAN-specific error
02	NOTIMP: not implemented
03	IGNORED: ignored requested disposition[1]
04	IGNNOTDEL: ignored requested disposition, file not deleted[1]
17	SYNERRNAM: syntax error in NAMELIST input
18	TOOMANVAL: too many values for NAMELIST variable
19	INVREFVAR: invalid reference to variable
20	REWERR: REWIND error
21	DUPFILSPE: duplicate file specifications
22	INPRECTOO: input record too long
23	BACERR: BACKSPACE error
24	ENDDURREA: end-of-file during read
25	RECNUMOUT: record number outside range
26	OPEDEFREQ: OPEN or DEFINE FILE required
27	TOOMANREC: too many records in I/O statement
28	CLOERR: CLOSE error
29	FILNOTFOU: file not found
30	OPEFAI: open failure
31	MIXFILACC: mixed file access modes
32	INVLOGUNI: invalid logical unit number
33	ENDFILERR: ENDFILE error
34	UNIALROPE: unit already open
35	SEGRECFOR: segmented record format error
36	ATTACCNON: attempt to access non-existent record
37	INCRECLEN: inconsistent record length

[1] Errors 3 and 4 do not cause an ERR transfer and do not assign an IOSTAT value.

Copyright © 1989–1994 Thinking Machines Corporation

Table 10. Summary of run-time I/O status codes *(cont'd)*.

Error #	Message Text
38	ERRDURWRI: error during write
39	ERRDURREA: error during read
40	RECIO_OPE: recursive I/O operation
41	INSVIRMEM: insufficient virtual memory
42	NO_SUCDEV: no such device
43	FILNAMSPE: file name specification error
44	INCRECTYP: inconsistent record type
45	KEYVALERR: keyword value error
46	INCOPECLO: inconsistent `OPEN/CLOSE` parameters
47	WRIREAFIL: write to `READONLY` file
48	INVARGFOR: invalid argument for I/O library
49	INVKEYSPE: invalid key specification
50	INCKEYCHG: inconsistent or duplicate key
51	INCFILORG: inconsistent file organization
52	SPERECLOC: specified record locked
53	NO_CURREC: no current record
54	REWRITERR: `REWRITE` error
55	DELERR: `DELETE` error
56	UNLERR: `UNLOCK` error
57	FINERR: `FIND` error
59	LISIO_SYN: list-directed I/O syntax error[2]
60	INFFORLOO: infinite format loop
61	FORVARMIS: format/variable-type mismatch[2]
62	SYNERRFOR: syntax error in format
63	OUTCONERR: output conversion error[2,3]
64	INPCONERR: input conversion error[2]
66	OUTSTAOVE: output statement overflows record
67	INPSTAREQ: input statement requires too much data
68	VFEVALERR: variable format expression value error[2]

[2] If an `ERR=` specifier is present, the transfer of control is taken *after* completion of the I/O statement for errors 59, 61, 63, 64, and 68, leaving the file status and record position as it was before the error occurred. (Other I/O errors leave the file status and record position undefined.)

[3] If no `ERR=` specifier is provided when error 63 occurs, the field in error is filled with asterisks and the program continues after the error message is printed.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation