6. The Basic Direct File Organisation

6.1 Basic Direct Files

A BD file must always be assigned to a direct access device, and is treated as a string of bytes numbered sequentially from zero upwards. Records are accessed by supplying the starting byte number as key, together with the number of bytes to be read or written.

6.1.1 Record Format

The format of BD records is entirely under program control since you determine the length and starting byte of each record. This makes the basic direct file organisation particularly suitable for handling special file structures such as databases.

6.1.2 Specifying File Attributes

The attributes of a file, such as its unit-id, volume-id and file-id, are specified in its file definition (FD), coded in the data division. When creating a new file you specify the space to be allocated in the FD.

You must specify the name of the area which contains the record length. You can also define a key area to contain the starting byte number for use in random access READ and WRITE statements. Section 6.2 below describes those parts of the file definition which are specific to the basic direct file organisation, but the statements which are common to all organisations are defined in section 1.2.

6.1.3 File Processing Statements

Nine procedure division statements are provided to enable a BD file to be processed. They are:

OPEN	which must be executed prior to any other statement affecting the file;
READ	to read a record at random;
READ FIRST	to read the very first record in the file;
READ LAST	to read the very last record in the file;
READ NEXT	to read the next record during sequential processing;
READ PRIOR	to read the previous record during sequential processing;
WRITE	to update an existing record at random;
WRITE NEXT	to write a record during sequential processing;
CLOSE	to terminate processing of a file.

When any of the statements is executed a file condition, signalled by exception condition 2, may arise as explained in section 1.3.1. In addition, if OPTION ERROR or ON ERROR is specified in the FD, an exception condition will be generated should an irrecoverable I/O error occur, as explained in section 1.6. Therefore it is usual to follow each file processing statement with an

ON EXCEPTION statement. If you do not, and an exception condition arises, your program will be terminated in error.

6.1.4 Extent Boundary Checking

A write next pointer is not maintained for a basic direct file, and you may use WRITE NEXT, WRITE, READ FIRST, READ LAST, READ NEXT, READ PRIOR and READ statements to access information anywhere within the allocated file extent. However, should any of these statements attempt to read or write a record which is partially or wholly outside the file extent, the file boundary violation file condition will be signalled.

6.1.5 The ORGANISATION Statement

If a program uses the basic direct access method then the statement:

```
ORGANISATION OR$85 TYPE x EXTENSION 8
```

must be coded in the data division before the first FD or data declaration. The type, x, may be any value in the range 0 to 99. It is the organisation type that will be given to any files created by OPEN NEW, and which will appear in a directory listing. We recommend that users avoid using organisations 0 to 9 as these are reserved for use by Global System Manager. (For a more detailed explanation of the ORGANISATION statement, refer to section 1.2.)

6.2 The File Definition

The file definition for a basic direct file is coded in either working storage or the linkage section as follows:

```
FD filename ORGANISATION OR$85
[ASSIGN TO UNIT unit-id FILE file-id [VOLUME volume-id]]
[KEY IS keyname]
RECORD LENGTH IS length
[SIZE IS size]
[OPTION ERROR]
[ON ERROR intercept]
```

The FD establishes a special group data item, 88 bytes in length, whose name is filename. The quantities unit-id, file-id, volume-id, keyname, length and size appear as subordinate items within this group and can, if need be, be referred to by the application program.

If it is possible to specify unit-id, file-id or volume-id before the program executes then the quantity should be coded as a character string in quotes. If you can specify the size or length before the program executes, code the quantity as a numeric string. If any of these quantities is not known until run-time then a symbol must be coded for the quantity. This symbol will then label a level 02 item which the user program is responsible for initialising.

6.2.1 The Filename

The filename must be a symbol. It serves to label the file definition, as explained in section 1.2.1.

6.2.2 The ORGANISATION Clause

The ORGANISATION clause must be coded as shown. It indicates that the basic direct access method is to be used.

6.2.3 The ASSIGN Statement

Use of the ASSIGN statement, which is the same for any file organisation, is explained in section 1.2.

6.2.4 Optional Statement Placement

The KEY, RECORD LENGTH, SIZE, OPTION and ON ERROR statements may appear in any order following the ASSIGN statement. The RECORD LENGTH statement must be coded: the others are optional.

6.2.5 The KEY Statement

The KEY statement is only required if records are to be accessed at random. Keyname is specified as a symbol and the statement:

```
02 symbol PIC 9(9) COMP
```

is generated within the FD.

BDAM maintains the keyname field to contain the byte number of the start of the record last accessed. The byte number of the very first byte of the file counts as byte number zero.

The program must place the byte number of the start of the record it requires to access in the keyname field before executing a random READ or WRITE operation.

A successful READ NEXT or WRITE NEXT operation increments the keyname field by the length of the previous record accessed and then accesses the record thus identified: if the previous operation on the file was an OPEN then the first record is accessed.

A successful READ PRIOR decrements the keyname field by the length of the record to be read, and then accesses that record.

6.2.6 The RECORD LENGTH Statement

The RECORD LENGTH statement is always required. Length is specified as a symbol and the statement:

```
02 symbol PIC 9(4) COMP
```

is generated within the FD.

The program must place the length of the record to be read in the length field before executing a read or write statement. The field is set to zero when the file is opened, and is not altered by read and write operations.

6.2.7 The SIZE Statement

The SIZE statement is required when creating a new file or truncating an existing one. Its use, which is common to all file organisations, is explained in section 1.2.8.

6.2.8 The OPTION and ON ERROR Statements

OPTION ERROR should be coded only if you wish your program to regain control following an irrecoverable I/O error. ON ERROR should be coded if you wish to handle certain I/O errors

specially. The processing of these statements is common to all file organisations, and is described in section 1.6.

6.2.9 Additional Fields in the FD

A basic direct access FD contains three additional fields which can be accessed by including a redefinition of the FD, as in the following example:

```
01          BD REDEFINES filename
  03        FILLER      PIC X(44)
  03        BDLAB
    05      FILLER      PIC X(3)
    05      BDORG       PIC 9(2) COMP
    05      FILLER      PIC X(12)
    05      BDADE       PIC X(8)
    05      FILLER      PIC X(12)
```

The field BDLAB, the file label area, is a 36-byte area Containing all the information about the file which is preserved in its label, including the organisation type, file-id and size. It is written to the file label by a successful CLOSE, and is returned by a successful OPEN OLD or OPEN SHARED. A successful OPEN NEW statement may corrupt certain parts of the label area.

If you use the basic direct access method in conjunction with a link handler constructed using Assembler Interface to transmit files of any organisation across a communications link then, in addition to the data, the label area should also be transmitted.

You should then use the basic direct access method to write the transmitted data to the direct access device. The file label area, containing all the access method dependent information, can then be written to the file label by moving it to the output file label area in the BD FD before closing the file. In this way it is possible to transmit files of any organisation, including relative sequential, indexed sequential and program files. There is a detailed example illustrating the technique in Appendix D, which contains a short program capable of copying files of any organisation using the basic direct access method.

Following a successful OPEN OLD or SHARED operation the field BDORG will contain the organisation type of the file opened, as specified in the ORGANISATION statement associated with the FD. This can be checked by your program to ensure that the organisation is as expected.

The field BDADE, the access method dependent area, is an 8-byte area which is available to the user program. It is written to the file label by a successful CLOSE statement, and is returned by a successful OPEN OLD or SHARED. Following an OPEN NEW operation the area will be set to binary zeros.

6.3 The OPEN Statement

The OPEN statement is coded:

```
OPEN type filename
```

where *type* is the word NEW, OLD or SHARED and *filename* identifies the basic direct file definition. An OPEN statement must be executed before any other operation affecting the file. If an OPEN is attempted but the FD is already open your program will be terminated in error.

OPEN NEW is used to create a BD file. OPEN OLD obtains exclusive access to an existing file and OPEN SHARED allows co-operating jobs running under multi-user Global System Manager to share a file. (The features of the open operation which are common to all file organisations, such as volume-id checking, are described in detail in section 1.3.2.)

6.3.1 File Conditions

The file already exists condition will be signalled in response to OPEN NEW if a file with the same file-id as that specified in the FD is already present on the direct access volume.

When an OPEN OLD or OPEN SHARED statement is executed Global System Manager checks to see whether a file with the same file-id as that specified in the FD is present on the volume, and that it is not a product file. If this is not the case, file not found is signalled. (Note that the organisation is not checked as is normally the case. If you wish to test that the organisation is as expected, examine BDORG as explained in 6.2.9.)

6.3.2 Successful Completion

Providing no file condition or irrecoverable I/O error occurs an OPEN NEW results in Global System Manager allocating the amount of space indicated by the FD's SIZE statement (or its absence).

When an OPEN OLD or OPEN SHARED statement completes successfully the file size is returned in the FD and can be accessed by the user program if a SIZE statement with the symbol option was coded. In addition the file organisation type and access method dependent area are returned in BDORG and BDADE, as explained in 6.2.9.

The record length field in the FD is set to zero by a successful OPEN NEW statement. When an OPEN OLD or OPEN SHARED statement completes successfully the record length returned is the value it had when the file was last closed. If the file is an IS or RS file this will be the actual record length of the file.

The keyname field in the FD is always set to zero following a successful OPEN statement. This is to allow the file to be processed sequentially using a sequence of READ NEXT or WRITE NEXT statements as described below.

6.3.3 Programming Notes

Since the basic direct access method does not check the file organisation it can be used to access any file, except a product file such as Global System Manager command library. You should not, however, attempt to update a file created by a different access method, as this is likely to result in the file becoming corrupt.

6.4 The WRITE NEXT Statement

WRITE NEXT is used to write all or part of a file sequentially. It is coded:

```
WRITE NEXT filename FROM A
```

Here *filename* identifies the basic direct file definition and A is a simple or indexed variable. If a WRITE NEXT is attempted on an FD which is not already open the program will be terminated in error.

6.4.1 Establishing the Length

The FD must contain the RECORD LENGTH statement with the length coded as a symbol. The program must establish the length of the record to be written (1 to 32767 bytes) in this field before executing the WRITE NEXT.

If the length is not established explicitly by the program the length of the last record accessed will be used.

6.4.2 File Conditions

An file boundary violation condition will be signalled if WRITE NEXT attempts to output a record which is wholly or partially outside the file extent.

6.4.3 Successful Completion

Provided that no file condition or irrecoverable I/O error occurs WRITE NEXT will set the key to the number of the byte following the previous record accessed (if any) and transfer A to the record thus identified. The number of bytes transferred is given by the length field.

6.4.4 Programming Note

If a KEY statement was not specified in the file definition the FD still contains an internal key field which is set to zero when the file is opened. In this case WRITE NEXT can be used to write the entire file sequentially.

When a KEY statement is specified a READ or WRITE can be used to position the file at any point. WRITE NEXT will continue writing from the record following the one accessed by READ or WRITE.

The READ NEXT statement sets the key field to the record retrieved, so that a subsequent WRITE NEXT statement will write the following record.

6.5 The WRITE Statement

WRITE is used to write a record at random or rewrite the last record accessed. It is coded:

```
WRITE filename FROM A
```

Here *filename* identifies the basic direct file definition and A is a simple or indexed variable. If a WRITE is attempted on an FD which is not already open the program will be terminated in error.

6.5.1 Establishing the Key

If WRITE is to be used as a random access operation the FD must contain a KEY statement coded as:

```
KEY IS symbol
```

which causes:

```
02 symbol PIC 9(9) COMP
```

to be generated. The program must then move the byte number of the start of the record it is required to retrieve to this field before executing the WRITE statement.

If a KEY statement is not coded in the file definition the FD still contains an internal key field which is set to zero when the file is opened and incremented by READ NEXT and WRITE NEXT. In this case the only use of the WRITE statement is to rewrite the record previously retrieved by READ NEXT or written by WRITE NEXT.

6.5.2 Establishing the Length

The FD must contain the RECORD LENGTH statement with the length coded as a symbol. The program must establish the length of the record to be written (1 to 32767 bytes) in this field before executing the WRITE.

If the length is not established explicitly by the program the length of the last record accessed will be used.

6.5.3 File Conditions

An extent boundary violation condition will be signalled if WRITE attempts to output a record which is wholly or partially outside the file extent.

6.5.4 Successful Completion

Provided that no file condition or irrecoverable I/O error occurs WRITE will transfer the number of bytes given by the record length from A to the record identified by the key.

6.5.5 Programming Note

A WRITE operation will always complete the updating of the file before returning control, so that even if the program subsequently fails the file will have been updated.

6.6 The READ FIRST and READ LAST Statements

READ FIRST is used to read the very first record of the file. It is coded:

```
READ FIRST filename INTO A
```

Similarly READ LAST is used to read the very last record in the file. It is coded:

```
READ LAST filename INTO A
```

In both cases *filename* identifies the basic direct file definition and A is a simple or indexed variable. If READ FIRST or READ LAST is attempted on an FD which is not open the program will be terminated in error.

6.6.1 Establishing the Length

The FD must contain the RECORD LENGTH statement with the length coded as a symbol. The program must establish the length of the record to be read (1 to 32767 bytes) in this field before executing a READ FIRST or READ LAST.

If the length is not established explicitly by the program the length of the last record accessed will be used.

6.6.2 File Conditions

A file boundary violation condition will be signalled if a READ FIRST or READ LAST attempts to input a record which is larger then the total file size, and which would therefore start or end outside the file extent.

6.6.3 Successful Completion

Provided that no file condition or irrecoverable I/O error occurs READ FIRST will set the key to address the first byte of the file (a value of zero), and READ LAST will set the key to be the file extent size less the length of the record to be processed. The record thus identified will be transferred to A, the number of bytes transferred being given by the length field.

6.6.4 Programming Note

Use of READ FIRST is normally used to reposition at the start of the file prior to reading records sequentially using READ NEXT.

Use of READ LAST presupposes that the program has sufficient information about the file to be able to determine the length of the very last record (possibly because the length is fixed). It is important to note that it is the responsibility of the program to determine an appropriate record length for use by READ LAST, not that of the basic direct access method itself.

6.7 The READ NEXT and READ PRIOR Statements

READ NEXT and READ PRIOR are used to process part or all of a file sequentially. READ NEXT is used to read the next sequential record, and it is coded:

```
READ NEXT filename INTO A
```

READ PRIOR is used to read the previous sequential record. It is coded:

```
READ PRIOR filename INTO A
```

In both cases *filename* identifies the basic direct file definition and A is a simple or indexed variable. If a READ NEXT or READ PRIOR is attempted on an FD which is not already open the program will be terminated in error.

6.7.1 Establishing the Length

The FD must contain the RECORD LENGTH statement with the length coded as a symbol. The program must establish the length of the record to be read (1 to 32767 bytes) in this field before executing the READ NEXT or READ PRIOR.

If the length is not established explicitly by the program the length of the last record accessed will be used.

6.7.2 File Conditions

A file boundary violation condition will be signalled if READ NEXT or READ PRIOR attempts to input a record which is wholly or partially outside the file extent.

6.7.3 Successful Completion

Provided that no file condition or irrecoverable I/O error occurs READ NEXT will set the key to the number of the byte following the previous record accessed (if any), and READ PRIOR will set the key to the number of the byte record length bytes before the start of the previous record

accessed. The record thus identified is transferred to A. In both cases the number of bytes transferred is given by the length field.

6.7.4 Programming Note

If a KEY statement was not specified in the file definition the FD still contains an internal key field which is set to 0 when the file is opened. In this case READ NEXT can be used to read sequentially through the entire file; READ PRIOR at any point may be used to retrieve the preceding record; WRITE updates the last record thus retrieved; and READ rereads it. In addition READ FIRST and READ LAST may be used to position at either the start or end of the file.

When a KEY statement is specified a READ or WRITE can be used to position the file at any point. READ NEXT will continue sequential processing from the record following the one retrieved by READ or output by WRITE, and READ LAST will process sequentially from the record before it.

A WRITE NEXT statement sets the key field to the start of the record written, so that a subsequent READ NEXT statement will read the following record, and a READ PRIOR statement will read the previous record.

Note that when using READ PRIOR it is the responsibility of the program to determine the length of the record to be processed by some means (possibly because it has a fixed length), and not that of the basic direct access method itself.

6.8 The READ Statement

READ is used to retrieve a record at random or reread the last record accessed. It is coded:

```
READ filename INTO A
```

Here *filename* identifies the basic direct file definition and A is a simple or indexed variable. If READ is attempted on an FD which is not already open the program will be terminated in error.

6.8.1 Establishing the Key

If READ is to be used as a random access operation the FD must contain a KEY statement coded as:

```
KEY IS symbol
```

which causes:

```
02 symbol PIC 9(9) COMP
```

to be generated. The program must then move the byte number of the start of the record it is required to retrieve to this field before executing the READ statement.

If a KEY statement is not coded in the file definition the FD still contains an internal key field which is set to zero when the file is opened and incremented by READ NEXT and WRITE NEXT. In this case the only use of the READ statement is to reread the record previously retrieved by READ NEXT or written by WRITE NEXT.

6.8.2 Establishing the Length

The FD must contain the RECORD LENGTH statement with the length coded as a symbol. The program must establish the length of the record to be read (1 to 32767 bytes) in this field before executing the READ.

If the length is not established explicitly by the program the length of the last record accessed will be used.

6.8.3 File Conditions

A file boundary violation condition will be signalled if READ attempts to input a record which is wholly or partially outside the file extent.

6.8.4 Successful Completion

Providing no file condition or irrecoverable I/O error occurs, READ will transfer to A the number of bytes given by the record length of the record specified in the key field.

6.8.5 The READ PHYSICAL Statement

As the key of a BDAM file is a byte number, the READ PHYSICAL statement functions in exactly the same way as a READ.

6.9 The CLOSE Statement

CLOSE must be used to terminate the processing of a file. It is coded:

```
CLOSE filename [TRUNCATE|DELETE]
```

where *filename* identifies the basic direct file definition.

6.9.1 Standard Processing

CLOSE always completes any outstanding I/O operations on the file and returns the FD to the status it possessed prior to being opened. Following CLOSE, the FD can be re-opened for the same (or a different) file by a subsequent OPEN NEW or OPEN OLD statement.

6.9.2 File Conditions

If the FD passed to the CLOSE was not open, then a file not open condition will be signalled.

6.9.3 Truncation

The TRUNCATE phrase is used in conjunction with a SIZE statement with the symbol option. The program must have established the size to which the file is to be truncated in the size field. When a CLOSE TRUNCATE is subsequently executed the additional part of the file, if any, will be returned to the volume so that it can be reallocated.

If you have created a new file using a sequence of WRITE NEXT Statements and wish to return any unused space to Global System Manager you must have specified the key field using the symbol option. To find the size of the file you have created simply add the record length of the last record output by WRITE NEXT and the current value of the keyname field together. Store this value in the size field, then CLOSE TRUNCATE the file.

Note that if a CLOSE TRUNCATE is attempted and the size specified is greater than the file's extent, the program will be terminated in error.

If you use the basic direct access method on a file created by a different access method, you must never truncate the file under any circumstances, except to set it to the current file size, as given by the FSTAT$ system routine.

6.9.4 Deletion

If a DELETE phrase is coded all the space the file occupies is returned to the volume and its file-id is destroyed. Following a CLOSE DELETE the file no longer exists.

6.9.5 Programming Notes

If you fail to close a basic direct file, records created by WRITE NEXT statements may not have been written to the file, as such records are buffered for efficiency.

A WRITE operation is effected immediately so that a file which is only updated by WRITEs should normally remain consistent, even if it is not closed. This feature is intended to protect files which are updated interactively from damage in the event of machine failure: programs should always close the files they use.

A CLOSE statement will write the access method dependent area in the FD (defined in 6.2.9) to the file label, so that it can be retrieved by a subsequent OPEN OLD or OPEN SHARED.